UNIVERSIDAD CARLOS III DE MADRID

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

DEPARTAMENTO DE INFORMÁTICA

TRABAJO DE FIN DE GRADO:

EXTRACCIÓN DE LA INFORMACIÓN PRESENTE EN DIAGRAMAS EN FORMATO IMAGEN

AUTOR: **EDUARDO DE TENA MARTÍNEZ**

TUTOR: **VALENTIN MORENO PELAYO**

SEPTIEMBRE 2016, LEGANÉS

EXTRACCIÓN DE LA INFORMACIÓN PRESENTE EN DIAGRAMAS EN FORMATO IMAGEN

A mis padres

UNIVERSIDAD CARLOS III DE MADRID

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

El tribunal aprueba el TFG titulado:

"EXTRACCIÓN DE LA INFORMACIÓN PRESENTE EN DIAGRAMAS EN FORMATO IMAGEN"

Tribunal:

Presidente:

ZUMEL VAQUERO, PABLO

Secretaria:

ROJAS DELGADO, BRENDA CAROLINA

Vocal:

MARTIN RUANO, DANIEL

Suplente:

COLORADO HERAS, EDUARDO

ÍNDICE

RESUMEN

La información forma parte de nuestra vida diaria. Desde que apareció internet, la cantidad que existe sobre cualquier materia que imaginemos es impresionante, y toda ella a un solo “click”: abrimos nuestro navegador de internet, escribimos la materia a buscar e inmediatamente encontramos a nuestra disposición más información de la que realmente somos capaces de usar.

Esta información se nos puede presentar en diferentes formas, desde un simple texto a un video o una imagen. Una de las dificultades a la que nos enfrentamos ya sea a nivel usuario o nivel experto es, de entre toda esa información disponible, saber cuál escoger y, sobre todo, saber cómo interpretarla.

Llegados a este punto, cada vez son más necesarios softwares capaces de realizar ese trabajo por nosotros debido a la creciente cantidad de información. Estos softwares deberán estar orientados a un tipo de materia concreta y a un formato de presentación concreto; y su eficiencia se medirá por la rapidez y exactitud con que sea capaz de analizar la información sobre la que trabaje.

El proyecto sobre el que trata esta memoria se centra en la información presente en **esquemas de circuitos eléctricos** en **formato imagen**. Para ello se ha creado un software que permite, una vez introducida una imagen, saber si dicha imagen es un circuito eléctrico para su posterior interpretación. De esta manera el usuario sabrá si una imagen es válida y sabrá luego que información encierra, evitando el tiempo que necesitaría para analizarla por su cuenta.

Debido a que este es el primer software orientado a analizar la información de esquemas de circuitos eléctricos en formato imagen, y a la gran variedad de formatos en los que se presentan dichos esquemas, sólo se trabajará sobre imágenes extraídas de la app “iCircuit”. La interfaz del software se ha realizado en Microsoft Visual Studio, usando las librerías Open CV.

Por ello, este software debe considerarse como punto de partida y no como un proyecto finalizado, ya que son muchas las alternativas para su implementación así como sus futuras aplicaciones.

Palabras clave: *Extracción de información, Microsoft Visual Studio, Librería OpenCV, app iCircuit, Circuitos electrónicos, Diagramas, Imagen.*

ABSTRACT

Information is part of our daily life. Since appearing online, the amount that exists on any matter that we can possibly imagine is impressive, and with a single "click" we can open our web browser, type the subject to search and immediately find to our disposal more information than we are really able to use.

This information we can present comes in different forms, from simple text to a video or an image. One of the difficulties we face, both user-level or expert level is, of all the information available, knowing which to choose and, above all, know how to interpret it.

To this this point, software is becoming increasingly necessary in order to be able to do this work for us because of the ever increasing volume of information. These software's should focus on a specific type of material and a specific presentation format; and efficiency will be measured by the speed and accuracy with which it is able to analyse the information on which to work.

The project on which this report is focused on is the information found within electrical plans in image format. For this software has been created that allows, once an image is inserted, whether that image is an electrical circuit for subsequent interpretation. This way the user will know if an image is valid, then know that information contains, avoiding the time one needs to analyse.

Because this is the first software of its kind aimed at analysing information from electrical plans in image format, and the variety of formats in which such schemes are presented, only images taken from the app "iCircuit" will be worked on. The software interface has been made in Microsoft Visual Studio, using the Open CV resources.

Therefore, this software should be considered as a starting point and not as a finished project, since there are many alternatives for implementation as well as future applications.

Keywords: *Information extraction, Microsoft Visual Studio, OpenCV library, app iCircuit, Electronic circuits, Diagram, Image.*

CAPÍTULO 1. INTRODUCCIÓN

Actualmente en la sociedad, uno de los cimientos básicos es el análisis de la información. Cada día la información presente en nuestras vidas, gracias entre otras cosas a internet, es mayor. Al estar permanentemente "conectados" tenemos acceso a muchísima información, por lo que no resulta difícil sentirse abrumado y perderse ante ella.

En un mundo, donde cada vez tenemos más ocupaciones, el tiempo se ha vuelto de vital importancia. Por ello, las aplicaciones que nos ayuden a optimizar nuestro tiempo son paulatinamente más necesarias y por lo tanto, más valoradas.

A ello se suma el desarrollo cada vez mayor de la tecnología, fundamentándose ésta en la ingeniería eléctrica y electrónica. Ambas se basan en el uso de circuitos eléctricos y electrónicos, siendo su punto común el uso de los diagramas que representan dichos circuitos.

El software que se ha implementado, busca resolver estos dos puntos: identificar los elementos que aparezcan en un circuito electrónico y ser capaz de analizar cómo están conectados. La idea es que la aplicación sea capaz de reconocer e interpretar todos los elementos posibles y, si apareciese alguno que no supiese interpretar, al menos indicar que hay un elemento desconocido.

La aplicación debe estar orientada tanto a profesionales del sector (ingenieros, académicos,…etc.) como a iniciados en esta materia (estudiantes,…etc.); y debe mostrar la información de manera clara y concisa. De esta manera, cualquiera podría usar e interpretar un circuito electrónico.

1.1. OBJETIVO

El propósito de este proyecto ha surgido debido a la ya incipiente necesidad de interpretar diagramas de circuitos electrónicos, sea cual sea la formación en dicha materia. Inicialmente, el proyecto se va a centrar en personas sin previa formación en circuitos electrónicos, como estudiantes, para que sean capaces de interpretar los diagramas ante ellos presentados.

El trabajo ha partido de cero en todo momento, es decir, sin la posibilidad de encontrar ninguna referencia sobre análisis de circuitos electrónicos mediante visión artificial. Por tanto, ya que hay numerosos programas de creación de estos circuitos, y todos diferentes entre sí, el proyecto se ha centrado en un solo programa: **iCircuit**, de **Apple**.

Por ello, el objetivo principal es desarrollar una aplicación para la extracción de **información** presente en **imágenes** de **esquemas de circuitos electrónicos** de la app **iCircuit**, con el fin de poder interpretar una serie de circuitos básicos sin necesidad de poseer conocimiento alguno sobre la materia.

Como objetivo secundario, crear una metodología aplicable a otros formatos de esquemas electrónicos, variando solamente ciertos parámetros concretos, pero manteniendo la estructura del programa. Las imágenes sobre las que se trabajará son vectoriales. La metodología incorporará técnicas de procesamiento de imágenes.

La interfaz va a dar la opción de introducir imágenes de circuitos electrónicos que previamente han sido descargados de la página oficial de iCircuit y luego va a mostrar los resultados obtenidos del análisis efectuado sobre la imagen seleccionada. Tiene que mostrar qué elementos aparecen en el circuito, así como el número total de ellos y sus conexiones.

1.2. MOTIVACIÓN

A lo largo de estos años de estudio de mi carrera universitaria, he constatado que las aplicaciones de diseño han tenido una gran relevancia. En casi todas las asignaturas cursadas, ha sido necesario el uso de estos programas para crear diagramas de apoyo al estudio, como en *Mecánica de estructuras* o *Fundamentos de la ingeniería electrónica*, por poner dos ejemplos.

Era usual compartir los diagramas creados entre compañeros a la hora de estudiar o realizar las prácticas de cada asignatura; estos diagramas los compartíamos exportándolos desde las aplicaciones. Uno de los problemas a los que nos enfrentábamos era que, en numerosas ocasiones, el compañero que recibía la información, no poseía la aplicación desde donde fue creada, por lo que únicamente podía verlo en formato imagen. Obtenida la imagen del diagrama, el problema era interpretarla, ya que para muchos elementos que aparecían, se carecía del conocimiento para saber qué eran.

Fue entonces cuando pensé que una aplicación que pudiese interpretar esos diagramas sería de gran utilidad, y que la elección de *diagramas de circuitos electrónicos* era sencilla, teniendo en cuenta que han sido los principales protagonistas de mis estudios.

El siguiente paso era plantear sobre que aplicación de diseño de circuitos electrónicos debía partir. En este caso, seleccioné la aplicación de **Apple** llamada **iCircuit**, por ser con la que he trabajado durante estos últimos años. De esta forma, cualquiera podría interpretar los circuitos creados por iCircuit sin necesidad de tener descargada la aplicación, y usar esa información como considerase.

Inmerso en esta idea, el trabajo surge como la necesidad de disponer de una aplicación que ayude a interpretar y compartir la información a partir de una imagen y tratar de abordar el problema planteado de la forma más óptima posible, siendo consciente de los numerosos retos a los que habrá que enfrentarse y con la esperanza de que este proyecto abra nuevas líneas de investigación.

1.1. ESTRUCTURA DE LA MEMORIA

La memoria está estructurada en **ocho capítulos**, perfectamente diferenciados, en los que son detallados desde el punto de partida teórico al desarrollo y conclusiones del proyecto. También se incluyen algunos anexos con información relevante al entendimiento del conjunto de capítulos.

El **capítulo primero** se centra en explicar el objetivo del proyecto, tanto el principal como el secundario. Además, se explica la necesidad del proyecto.

El **capítulo segundo** se centra en el estado del arte implícito en el desarrollo del proyecto. Comienza con el concepto de información y cómo se ha ido desarrollando a lo largo de la historia, tanto en el ámbito de su sentido en la sociedad como en el ámbito de las plataformas usadas para su almacenamiento. Esta parte es importante, ya que este proyecto se basa en la extracción e interpretación de la información de diagramas de circuitos eléctricos, así que se hacía necesario saber sobre qué se está trabajando. También se analiza cómo están formadas las imágenes digitales, sobre las cuales se extraerá la información. Debido a que es la herramienta que se usará para la consecución de los objetivos, se hace de crucial importancia saber qué es la visión artificial y cuáles son sus principales componentes. Se adelantarán también los algoritmos implementados en el proyecto, de manera teórica.

El **capítulo tercero** tratará sobre la descripción general, en el que se analizará de manera somera los elementos sobre los que se ha realizado este proyecto. Dicho de otra forma, se explicará el entorno de programación utilizado y las plataformas que han servido de apoyo.

El **capítulo cuarto** será el núcleo principal de la memoria, al tratarse éste del desarrollo del proyecto. Se explicarán todos los pasos que se han seguido para obtener los objetivos; quedarán explicados todos los procedimientos implementados dentro del programa. Se introducirá un manual de usuario para el uso del programa, intentando que sea lo más sencillo de usar para el usuario. Por último, aparecerán una serie de tablas

con los resultados de las pruebas realizadas con el programa, reflejando los aciertos y fallos e intentando analizar el porqué.

El **capítulo quinto** contiene los resultados obtenidos después de aplicar todos los cambios del programa. Se mostrará algún ejemplo y unos ratios de acierto la aplicación.

El **capítulo sexto** recogerá las conclusiones del proyecto, así como los trabajos futuros que puedan generarse a partir de él.

El **capítulo séptimo** incluye el presupuesto y el marco legal.

El **capítulo octavo** reflejará la bibliografía que se ha usado.

A partir de aquí y hasta el final de la memoria, aparecerán varios anexos en los que se detalla cómo instalar los componentes del proyecto y varios ejemplos de la aplicación en su forma final.

CAPÍTULO 2. ESTADO DEL ARTE

2.1. INFORMACION

2.1.1. Concepto

La información se define como un conjunto de datos procesados y ordenados para su comprensión, aportando nuevos conocimientos a un individuo o sistemas acerca de una materia, fenómeno o ente concreto. En consecuencia nos permite resolver problemas, tomar decisiones o buscar alternativas que se adapten mejor a nuestras necesidades, por lo cual, su uso razonable es el fundamento del conocimiento (1).

A su vez proporciona razón o soporte al entorno que percibimos, ya que, tomando como herramienta grupos de datos y reglas, da forma a diferentes arquetipos del intelecto humano.

Dada su importancia, la información debe tener relevancia, utilidad, vigencia y ser confiable, pues de ello dependen acciones o decisiones que puedan ser tomadas.

2.1.1 Evolución histórica: Cronología

El concepto de información ha ido cambiando a lo largo de nuestra existencia. Desde que los primeros hombres plasmaran cómo veían el mundo sobre las paredes de sus cuevas, dando lugar a las famosas pinturas rupestres, todos ellos han entendido desde su perspectiva qué mensaje era importante transmitir, así como la técnica necesaria para ello.

Aunque el concepto de información en sí mismo ha permanecido inmutable a lo largo del tiempo, no ha sido así en su contenido y transmisión. Al igual que ha variado el modo de entender su almacenamiento, pues de poco sirve transmitir un mensaje si no se conserva en una plataforma.

La plataforma de almacenaje de información más antigua es el propio ser humano: su memoria. En un principio toda la información que

recopilaba el hombre era almacenada por éste, y transmitida de boca en boca. Pero tenía un importante fallo: la seguridad de su almacenaje (ejemplo: la información podría perderse si fallecía la persona y no lo había trasmitido). Por ello, se debía desarrollar alguna nueva plataforma de almacenaje y transmisión de la información.

A partir de esa idea de la búsqueda de nuevas formas de almacenar y transmitir la información, el ser humano ha ido desarrollando métodos de lo más diverso: uso de materiales como nudos, conchas, pinturas rupestres o marcas sobre piedras. Sin embargo el salto definitivo más efectivo a la hora de almacenar y transmitir la información es con la escritura (2).

-Cronología

La historia de la información pasa por diferentes épocas que se explicarán de manera somera a continuación (3):

Siglos V a X – Alta Edad Media: En las bibliotecas de los monasterios se realiza la transmisión y el almacenamiento. El acceso y uso está limitado a ciertas clases sociales.

Siglo XV – Edad Moderna: La invención de la imprenta (Gutenberg) en Europa, implica que al producir libros en serie, el acceso la información es mayor. Aparecen los primeros periódicos.

Siglo XX – Edad Contemporánea:

- 1926: Primera *retrasmisión de televisión* que tendrá un profundo impacto durante todo el siglo.
- 1940: Definición del término información desde una *perspectiva científica* (Jeremy Campbell) dentro de una incipiente era electrónica.
- 1943: Invención de la *radio*(Nikola Tesla)
- 1947: Invención del *transistor* (Walter Houser Brattain, John Bardeen y William Bradford Shockley).Se asienta el primer principio en la tecnología de la información, ya que se integra microelectrónica y cálculo del ordenador.

- 1948: Elaboración de los cimientos matemáticos de la *Teoría de la Información* aplicando el Álgebra de Boole (Claude E. Shannon), asentando segundo principio en la tecnología de la información y la comunicación; de esta forma se crea la *Computación o Informática*. Con el uso de la numeración binaria y el transistor el mundo entra en la *Era Digital.*
  En este año también, la idea de la *cibernética* es elaborada por Norbert Wiener.

- 1951 – 1953: Descubrimiento de los principios de los *códigos de ADN*, (James Watson y Francis Crick) y lo que implica este sofisticado sistema de información para su uso.

- 1969: Se crea la primera conexión de computadoras con fines militares: ARPANET (precursor de INTERNET).Su popularización y expansión será el comienzo de nuevas relaciones en un mundo más interdependiente.

Siglo XXI – Actualmente tenemos acceso a gran cantidad de información, con capacidad creciente de almacenamiento y en soportes cada vez más pequeños.

2.2. VISIÓN ARTIFICIAL

Se trata de una tecnología, cuyo fin es la obtención de información del entorno real a través de imágenes digitales, utilizando un ordenador. Se utiliza en: inteligencia artificial, robótica, procesado de señal, reconocimiento de formas, teoría de control, psicología, neurología,…, etc.

2.2.1. Imagen digital

Una *imagen* es la representación visual de un objeto (pintura, fotografía, video, etc.) (4).

En tecnología, la palabra *digital* se refiere a la representación de información de modo binario.

Por tanto una **imagen digital**: es una representación visual bidimensional constituida a partir de una matriz binaria (compuesta de unos y ceros).

La forma de conseguir una imagen digital puede ser por dispositivos de conversión analógica –digital (cámara fotográfica digital, escáner..), o utilizar en el ordenador programas de tratamiento de imágenes. Independientemente al dispositivo utilizado, almacenaremos en el ordenador esa imagen digital mediante **bits.**

2.2.1.1. El píxel

El pixel es la división más pequeña de color o gama de grises de una imagen digital que son iguales (5).

No tiene medida concreta, es decir, que la medida es dividir en celdillas una retícula.

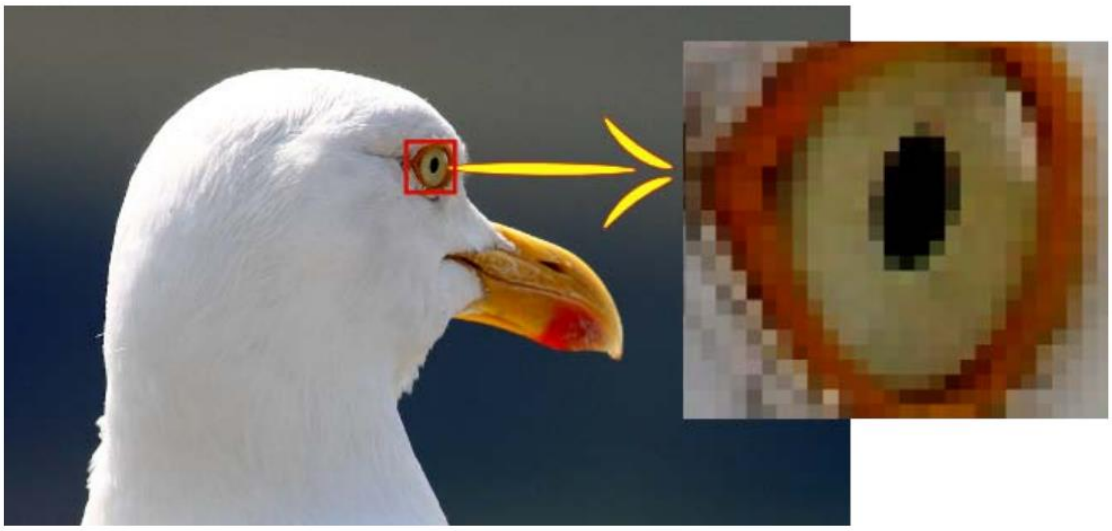


Figura 2.1: *Detalle de los pixeles dentro de una imagen.*

Si se observa el área señalada en rojo de la *Figura 2.1*, se aprecia el conjunto de pixeles que forman una imagen.

1. Profundidad de píxel

Cuando se habla de profundidad de un pixel, nos referimos a la cantidad de bits de información q se necesita para representar un color en una imagen digital, no a su posición espacial.

Al tratarse de un sistema binario de numeración, la profundidad de 1 bit equivale a que un pixel de la imagen puede tener dos valores (0 y1) es decir, dos colores diferentes: blanco o negro. A sí mismo, la profundidad de píxel de 8 bits implica que cada píxel pueda tener 256 colores distintos o 256 distintos de grises, ya que combinamos ceros y unos en series de 8 elementos.

En definitiva, la gama de colores de una imagen vendrá determinada por el número de bits por píxel. Por tanto, a mayor profundidad de bit, dispondremos de más colores y con mayor exactitud tendremos la representación del color en la imagen digital.

Así se refleja en la *Figura 2.2, Figura 2.3* y *Figura 2.4*: donde se muestra en la columna de la izquierda el color y número de bits, y en la columna de la derecha como resulta la imagen.

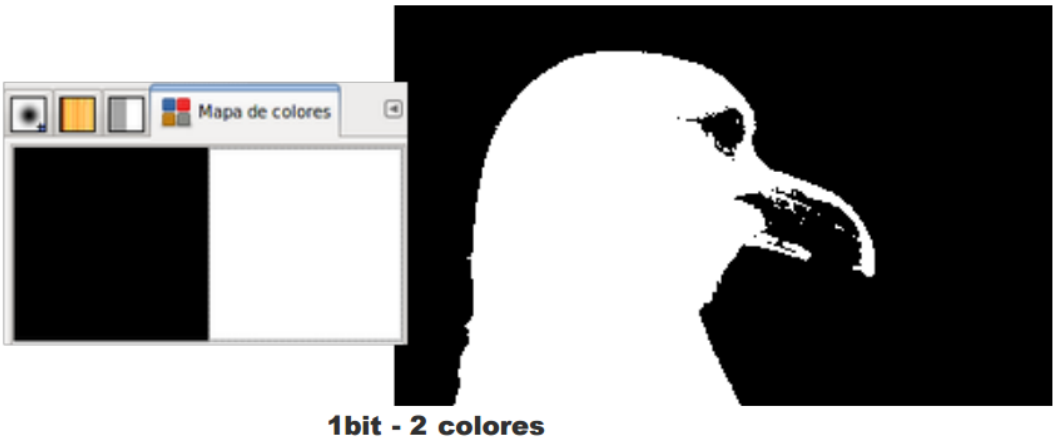


Figura 2.2: *Una imagen de 1 bit.*

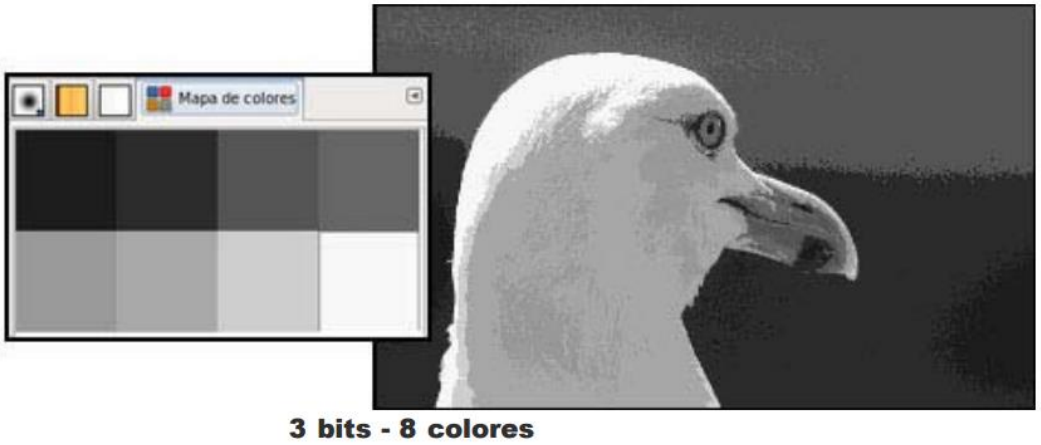


Figura 2.3*: Una imagen de 3 bit.*

Figura 2.4: *Una imagen de 6 bit.*

.

Figura 2.5*: Una imagen de 16 bit.*

.

Verdaderamente el "*color real"* se corresponde la última imagen de la *Figura 2.5.*

En una imagen de 256 colores, cada píxel podría coger uno de esos 256 colores. En consecuencia si la profundidad es de 24 bits, podemos manejar millones de colores con solo combinar los tres primarios: rojo (Red), verde (Green), y azul (Blue), conocido como **RGB.**

2. Resolución de una imagen

La resolución se determina por la cantidad de pixeles q forman una imagen de mapa de bits.

A mayor cantidad de pixeles por unidad de medida, mayor calidad tendrá la imagen. Esto implica que la calidad está vinculada con la resolución de una forma directa: a mas resolución más información y detalle en la calidad, y a menor resolución menor calidad de la imagen.

2.2.2. Etapas en un sistema de visión artificial

Las etapas de una aplicación de visión artificial se pueden dividir, según el grado de complejidad, en visión de bajo nivel y visión de alto nivel, que a su vez contienen diferentes etapas. Se pueden aplicar las etapas que se consideren, pero siempre y cuando se mantenga el orden de implementación que se describe a continuación (6).

- Visión de bajo nivel:
  - Adquisición y representación de la imagen
  - Preprocesamiento
  - Extracción de características
- Visión de alto nivel:
  - Segmentación
  - Transformaciones morfológicas
  - Descripción de objetos
  - Reconocimiento o clasificación

2.2.2.1. Breve análisis de las etapas más significativas

- Adquisición y representación de la imagen:

  La intensidad de cada componente de una base de color (RGB) en imágenes a color o el brillo en imágenes en niveles de gris se almacena en cada pixel.

- Preprocesamiento:

  Algoritmos que consiguen mejorar la imagen original, resaltando determinadas características de la imagen o eliminando otras que las ocultan. Algunos de estos algoritmos son: contraste, modificación del histograma, eliminación de ruido, realce de los bordes de la imagen o falso color.

- Segmentación:

  Para facilitar un reconocimiento o análisis automático (los límites de una palabra, buscar una cara etc.) dividimos una imagen en zonas homogéneas en función a una o más características (ejemplo: el

brillo, el color, contornos, texturas). De esta forma, cada pixel contiene una etiqueta que corresponde a un objeto, lo que da lugar a otra imagen. Algunos de los algoritmos son: textura, detección de contornos, crecimiento de regiones o movimiento.

- Transformaciones morfológicas:

  Son transformaciones que modifican la estructura o forma de los objetos que hay presentes en la imagen. Su utilidad es tanto la extracción de características como la eliminación de los errores que se producen en todo proceso de segmentación. Algunos de los algoritmos son: transformaciones hit or miss, erosión, dilatación, opening and closing, esqueletización, adelgazamiento, extracción de perímetro, eliminación de ruido, cerco convexo o eliminación de ramas.

- Descripción de objetos**:**

  Son aquellos algoritmos que se centrar en encontrar características destacables dentro de la imagen, con el fin de obtener posteriormente información relevante a través de ellas. Los algoritmos se pueden agrupar en dos grandes grupos:

  - **Características de la región:**
    - Tamaño, perímetro, compacticidad
    - Posición
    - Orientación
    - Momentos invariantes a la rotación, la escala y la traslación
  - **Características de la forma:**
    - Descriptores topológicos
    - Códigos encadenados
    - Signatura
    - Descriptores de Fourier

- Reconocimiento o clasificación:

  Partiendo de las cualidades encontradas y de los posibles objetos, se catalogan los que hay presentes en la imagen. Siendo la última etapa, se trata de un trabajo complejo y arduo y con frecuencia se simplifica para que funcione bien. Se suelen utilizar clasificadores, enfoques, reconocimientos probabilísticos, funciones discriminantes o algoritmos basados en distancias.

2.2.3. Algoritmos de visión artificial

Hoy día, disponemos de una gran variedad de técnicas para el tratamiento informático de imágenes; estos tratamientos se sustentan en la modificación de ciertos parámetros de aquellas, con el fin de ser capaces de extraer y analizar la información deseada (7).

En este proyecto se han utilizado algunas técnicas centradas en el reconocimiento de objetos, detección de bordes y segmentación. Las técnicas de visión artificial utilizadas son:

- Dilatación
- Detección de bordes
- Detección de contornos
- Drawing
- Bounding Boxes
- Región de interes

2.2.3.1. Análisis de los algoritmos implementados

- **Dilatación**

La dilatación (y erosión) es una de las operaciones fundamentales en el procesamiento morfológico de imágenes, a partir de la cual se basan todas las otras operaciones morfológicas. Originariamente, fue definida para imágenes binarias, para más tarde extenderse a imágenes en escala de

grises y posteriormente a retículos completos. La operación de dilatación normalmente usa un elemento estructural para expandir las formas contenidas en una imagen de entrada.

Ejemplo

Suponga una matriz A 11 x 11 y otra matriz B 3 x 3, como se aprecia en la *Figura 2.6*:

```
0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 0 0 1 1 1 0
0 1 1 1 1 0 0 1 1 1 0
0 1 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 1 0          1 1 1
0 1 1 0 0 0 1 1 1 1 0          1 1 1
0 1 1 0 0 0 1 1 1 1 0          1 1 1
0 1 1 0 0 0 1 1 1 1 0
0 1 1 1 1 1 1 1 0 0 0
0 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0
```

***Figura 2.6**: Matriz 11x11 y matriz 3x3.*

Para cada píxel en A, superponer el centro de B; la dilatación de A por B da una matriz 11 x 11 (*Figura 2.7*):

```
1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 0 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 0
1 1 1 1 1 1 1 1 1 0 0
```

***Figura 2.7**: Matriz resultado.*

- **Detección de bordes**

La detección de bordes nos permite averiguar los contornos de una imagen, o mejor dicho, los cambios bruscos de intensidad lumínica. Los algoritmos orientados a los bordes tratan de extraer los objetos de la imagen localizando sus contornos o fronteras. Generan como salida una imagen denominada *mapa de bordes*; éste puede incluir información explícita sobre: la posición, la fuerza o intensidad, o la orientación.

El detector de bordes más utilizado es el Detector de Canny, ya que optimiza una serie de condiciones.

- Condiciones:

- Detección: Evita la supresión de bordes importantes y no proporcionar bordes falsos.

- Localización: Constata que la distancia entre la posición del borde localizado y el borde real sea mínima.

- Respuesta: Que la respuesta sea mínima, es decir, identificar una sola vez un borde marcado, para ello debe agrupar las múltiples respuestas y minimizar el ruido para no propiciar bordes falsos.

- Los pasos para obtener el mapa de bordes con Canny es el siguiente:

- Filtro gaussiano y obtención del gradiente; la derivada de una gaussiana es el mejor operador para calcular la orientación y valor del vector (*Figura 2.8*).

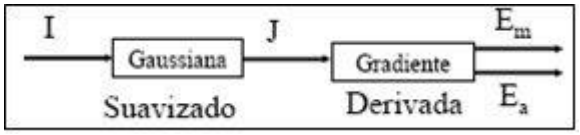


Figura 2.8: *Diagrama de la obtención del gradiente.*

- Eliminación de no máximos al resultado del gradiente: Con el gradiente obtenido, alcanzamos bordes de un pixel de ancho, despreciando aquellos que no lleguen a ese máximo en ese valor.

- Histéresis de umbral: Para reducir la aparición de contornos falsos, ruidos etc., se fija umbral, filtro por ***p*** (Primario) y añado por ***q*** (nivel bajo).

- Cerrar los contornos abiertos: mediante el algoritmo de Cocquerez y Deriche

- Resultado

Figura 2.9: *Imagen de la que se han obtenido los bordes.*

- **Detección de contornos**

Un contorno es una curva que une todos los puntos continuos (a lo largo del límite), siempre con el mismo color o intensidad. Los contornos son muy útiles para detectar y reconocer objetos, así como para el estudio de las formas.

Para una mejor precisión, se deben usar imágenes binarias; así que antes de encontrar contornos, es aconsejable aplicar en detector de bordes Canny.

- **Drawing**

Las funciones de *drawing* (o traducido al español, "*dibujando"*) trabajan con matrices (imágenes) de profundidad arbitraria. Su objetivo es mostrar sobre la imagen un conjunto de píxeles con el color deseado y la forma que se quiera describir; tan sólo tienes que indicar las coordenadas que se unirán para formar el "dibujo", así como los valores del color RGB.

Ejemplo

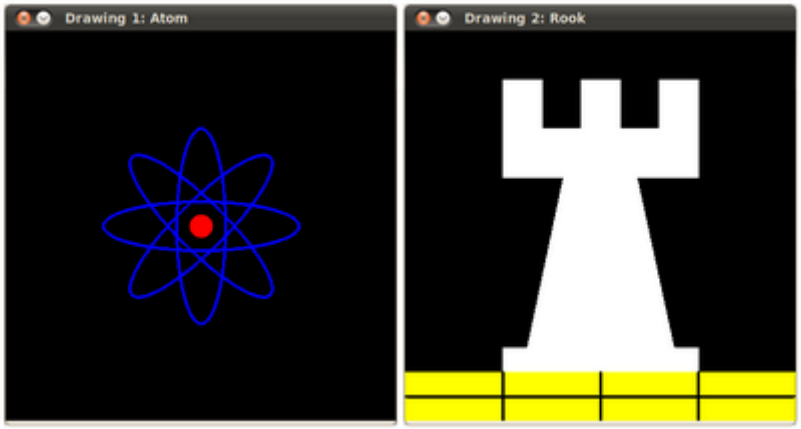


Figura 2.10: *Imagen realizada con la función drawing.*

En la *Figura 2.10,* se aprecian unas formas realizadas con los algoritmos de la función *drawing*.

- **Bounding Boxes**

Se trata de unos algoritmos que permiten "encerrar" los objetos de la imagen en una forma determinada, lo que permite entre otras cosas, calcular cuántos objetos hay en la imagen. Las formas para encerrar a los objetos pueden ser muy variadas, dependiendo de si se utiliza las formas prefijadas o se programa las propias. Una de las más usadas (y las que he usado en este proyecto) es el rectángulo. Este rectángulo lo reproduce según cuatro coordenadas, calculadas a partir de los límites del objeto a

encerrar, estando las líneas del rectángulo pegando a los bordes del objeto.

Ejemplo

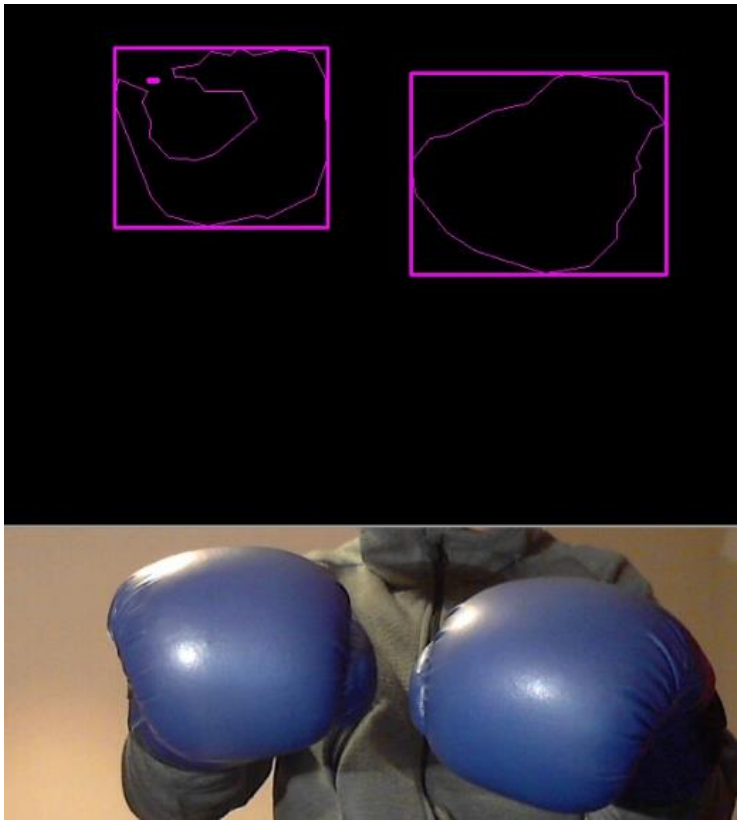


Figura 2.11: *Obtención de los Bounding Boxes.*

- **Región de interés**

Se trata de un algoritmo mediante el cual se extrae de la imagen una región de interés, bien para mostrarla bien para trabajar con ella. Es muy útil cuando se necesita sólo una parte concreta de la imagen.

Tan solo es necesario de unos pocos parámetros, como el punto desde el que se extraerá la región de interés y la altura y anchura de esa región.

CAPÍTULO 3. DESCRIPCIÓN GENERAL

Como se ha ido comentando en los apartados anteriores, este proyecto propone desarrollar una aplicación capaz de extraer la información encerrada en diagramas de circuitos eléctricos iCircuit de Apple, dividiéndose ese objetivo en distintos objetivos intermedios, como interpretar cuántos elementos electrónicos hay en el circuito o dónde están situados.

A continuación, se detallarán las herramientas utilizadas en este proyecto, desde el entorno del software hasta las librerías exactas que se han usado.

3.1. ENTORNO DE PROGRAMACIÓN

3.1.1. .NET

El *Framework de .Net* es una solución que ofrece Microsoft (libre y de código abierto) a un problema de programación, como es facilitar un código. Se trata de un equipamiento que reúne un conjunto de lenguajes - C#(C Sharp), Visual Basic o C++ h, Perl o Cobol etc- y servicios para poder generar aplicaciones (8).

Brinda un ámbito de ejecución enormemente organizado, permitiendo producir aplicaciones escalables y firmes, de tal forma que aunque cada lenguaje tiene sus propias características, con .NET se pude desarrollar cualquier tipo de aplicación.

Algunos de sus elementos son:

- Biblioteca de clases de .NET
- CLR (Common Languaje Runtime)
- Lenguajes de compilación

- **Arquitectura de .Net Framework**

Para poder aprovechar las características de esta plataforma se necesita comprender su estructura.

Arquitectura de .Net Framework

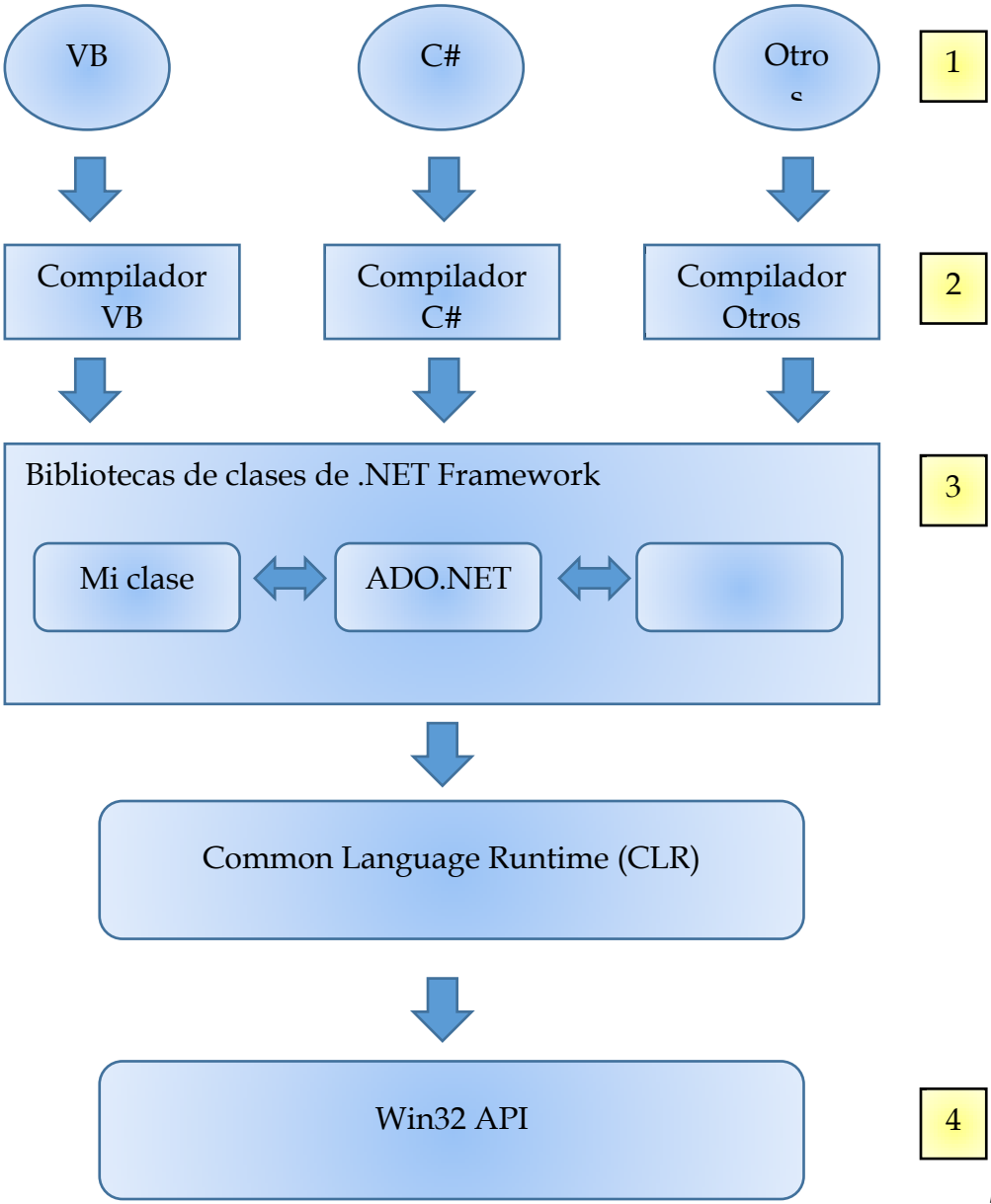


Figura 3.1: *Diagrama de flujo de la Arquitectura .N.*

En la *Figura 3.1* se refleja el esquema del funcionamiento de .NET Framework. Las partes que contiene son:

- 1: Código fuente
- 2: Compilador
- 3: Lenguaje Intermedio de Microsoft (MSIL)
- 4: Sistema Operativo

Common Language Runtime (CLR)

Podemos decir que el centro principal del Framework de .NET es, el CLR, siendo el marco de ejecución en el que se cargan las aplicaciones creadas en los diferentes lenguajes, aumentando el grupo de servicios que da el sistema operativo estándar Win32.

Compila el código fuente de lenguajes soportados por .NET en un mismo código, denominado *código intermedio* (MSIL, Microsoft Intermediate Lenguaje). Para crear ese código, el compilador se sustenta en el Common Language Specification (CLS) que define las pautas imprescindibles para proporcionar el código MSIL compatible con el CLR.

Como el MSIL genera automáticamente el lenguaje, el código siempre es el mismo, ahora bien al no ser un código máquina es necesario crear un segundo código para que pueda ser ejecutado en la plataforma que el ordenador posea: compilador JIT (Just-In-Time).

Así conseguimos que, partiendo del código MSIL, cada plataforma genere su código máquina y tenga su compilador JIT.

Según se van refiriendo los métodos en el programa, el CLR realiza la compilación JIT, almacenando en la memoria caché de la computadora el código ejecutable obtenido, solo recompilado si se produce una modificación en el código fuente.

Bibliotecas de clases de .NET

La biblioteca de clases de .NET está compuesta por espacios de nombres, donde cada espacio almacena tipos q se pueden emplear: enumeraciones, estructuras, interfaces y clases.

Agrupa las tecnologías Windows en un único escenario para los lenguajes de programación (ASP.NET, ADO.NET, Web Forms, GDI+, etc.)

Sus ventajas son:

- Su robusta estructuración
- Determinar tantos espacios de nombres como sean necesarios en el mismo proyecto.
- Si los espacios de nombres son desiguales, poder utilizar diferentes clases con el mismo nombre en un mismo programa.

Ensamblados

Con el fin de encontrar una solución para tratar con distintos archivos manejamos la idea del ensamblado.

Lo definimos como un conjunto lógico de varios ficheros o módulos (HTML, GIF, etc.) que se enmarcan bajo el mismo nombre, y que admite aislarse de la posición física de los recursos o el código.

Un ensamblado es una agrupación lógica de uno o más módulos o ficheros de recursos (ficheros .GIF, .HTML, etc.) que se engloban bajo un nombre común. Tienen forma de EXE o DLL y engloban la información imprescindible dentro de sí mismos, en lo que llamamos ***manifiesto del ensamblado.***

El manifiesto, por tanto, son meta –datos que recoge la información sobre sus peculiaridades (nombres de ficheros del ensamblado o de otros ensamblados, identidad, permisos de seguridad, dependencias etc.)

Hay dos tipos de ensamblado:

- Privados: Son ensamblados que se utilizan por una aplicación; lo designa el compilador por defecto.

- Públicos o compartidos: Ideados para ser utilizados por cualquier aplicación y que son almacenados en la *caché global*.

Sus ventajas son:

- Son auto-descriptivos, por lo que no necesita instalación.
- Versionado
- En lugar de registro a nivel de máquina, uso de ensamblados públicos

3.1.2. Microsoft Visual Studio 2015

Microsoft Visual Studio 2015 es un ámbito de mejora para desarrollar aplicaciones para Windows, Android e iOS, aplicaciones web, etc.

Esta nueva versión trae un gran número de cambios, entre ellos, que interactúa con otras plataformas, trabaja muy bien en Cloud Computing con Azure y no se vuelve inaccesible gracias a que .NET es libre y de código abierto (9).

Visual Studio facilita diferentes plantillas de aplicación para ayudar a crear programas y permite programar en varios lenguajes .Estos lenguajes de programación comparten similitudes, pero también poseen algunas diferencias y en ellas radica su elección para trabajar (10).

En este sentido y para el desarrollo de este proyecto, se ha elegido el lenguaje **Microsoft Visual C++** (MSVC ++) ya que su diseño permite controlar en detalle al compilar aplicaciones por medio de .NET Framework.

3.2. LIBRERIAS OpenCV

OpenCV es una biblioteca desarrollada por Intel, cuyo objetivo es ofrecer una infraestructura de visión por ordenador sencilla de utilizar que favorece desarrollar aplicaciones de CV (Computer visión) rápidamente (11).

Su diseño tiene una perspectiva hacia las aplicaciones de tiempo real y es eficiente en el gasto de medios informáticos.

Contiene una librería de aprendizaje automático muy completa, que facilita cualquier problema de adiestramiento.

Está escrita en C y C++, es compatible con GNU/Linux, Mac OS X y Windows (12). Engloban gran cantidad de áreas en el proceso de visión, gracias a sus múltiples funciones, como son la calibración de cámaras, vigilancia de video, visión estéreo, seguimiento y reconocimiento de objetos, análisis de imágenes o interfaces gráficos, como se ve en la *Figura 3.2*:

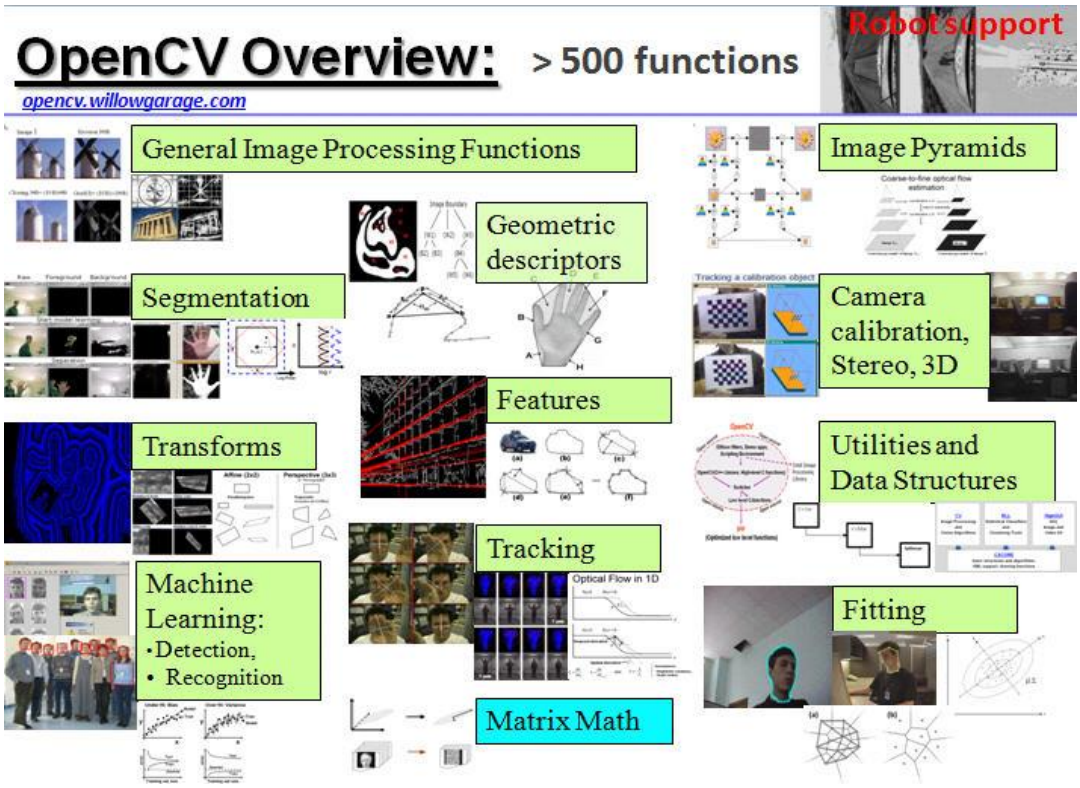


Figura 3.2: *Áreas y funciones del OpenCV.*

https://osl.ull.es/software-libre/opencv-libreria-vision-computador/

3.3. iCircuit

Aplicación por excelencia de iPad y iPhone para diseñar y experimentar con circuitos (13).

Puede manejar circuitos analógicos y digitales, gracias a su moderno mecanismo de simulación, realizando el análisis en tiempo real. Es la aplicación perfecta para profesionales, estudiantes o aficionados. (14)

Se utiliza igual que un programa de diseño y modelado tipo CAD: se añaden elementos, se conectan, y se ven sus propiedades.

Sin embargo iCircuit es diferente a otros programas tipo CAD porque está siempre simulando; es como hacerlo con un circuito real. El usuario no consume tiempo elaborando informes o tomando una medida, ya que el circuito se maneja como normalmente se haría, con la energía encendida.

La aplicación cuenta con más de treinta elementos para usar en los circuitos, desde simples resistencias, a interruptores, MOSFETS (transistores) o puertas digitales.

Tiene además, un multímetro que puede ser probado por todo el circuito e instantáneamente leer voltajes y corrientes. Si se quiere ver cómo oscila un valor en el tiempo, se puede añadir valores para construir el osciloscopio. La mira del osciloscopio puede leer simultáneamente varias señales a lo largo del tiempo y tiene una interfaz táctil para ver las características de las señales, pudiendo comparar varias de manera muy visual.

CAPÍTULO 4. DESARROLLO DEL PROGRAMA

El primer paso para desarrollar el programa de extracción de información es realizar la ***configuración del entorno***. Primero se deberá instalar el programa Visual Studio 2015 (en el momento del desarrollo es la última versión) en Windows 7, que es el sistema operativo que he elegido. Después, se descargará de la página web conveniente la librería OpenCV para su incorporación a un proyecto creado dentro de Visual Studio; y una vez configurados todos los parámetros, se creará, a partir de ese proyecto con las librerías ya incorporadas, una plantilla que permitirá crear proyectos de OpenCV sin necesidad de configurar todo ello desde el principio.

Una vez realizado todo lo anterior, ya está todo preparado para empezar a programar. Antes de empezar a escribir código, es necesario establecer un ***marco de trabajo*** apropiado, ya que intentar abarcar todo el posible potencial de trabajo del proyecto sería imposible. Una vez establecido el marco de trabajo, es de gran importancia detectar todos los *posibles inconvenientes* que pudiera tener ese espacio de trabajo; desde inconvenientes que no se sepan solucionar, hasta fallos que, incluso detectándolos, no se sepan resolver. Dependiendo de todos estos puntos, se crearán unas ***condiciones iniciales de funcionamiento*** del programa, las cuales deberán ser tomadas en cuenta para su análisis.

Ahora sí, se procederá a escribir el ***código del programa***. En todo momento se tratará de crear un programa base o estructura que permita más adelante añadir código para hacer más amplio y compacto el programa, pero manteniendo invariable dicha estructura.

4.1. MARCO DE TRABAJO

En este apartado se procederá a establecer un marco sobre el que trabajar.

4.1.1. Componentes analógicos

Inicialmente se acotará el espacio de estudio. Dado que en el mundo electrónico hay numerosos componentes o elementos, este proyecto se va a centrar en los principales **elementos electrónicos analógicos**. Estos componentes se pueden ver en la *Tabla 4.1*:

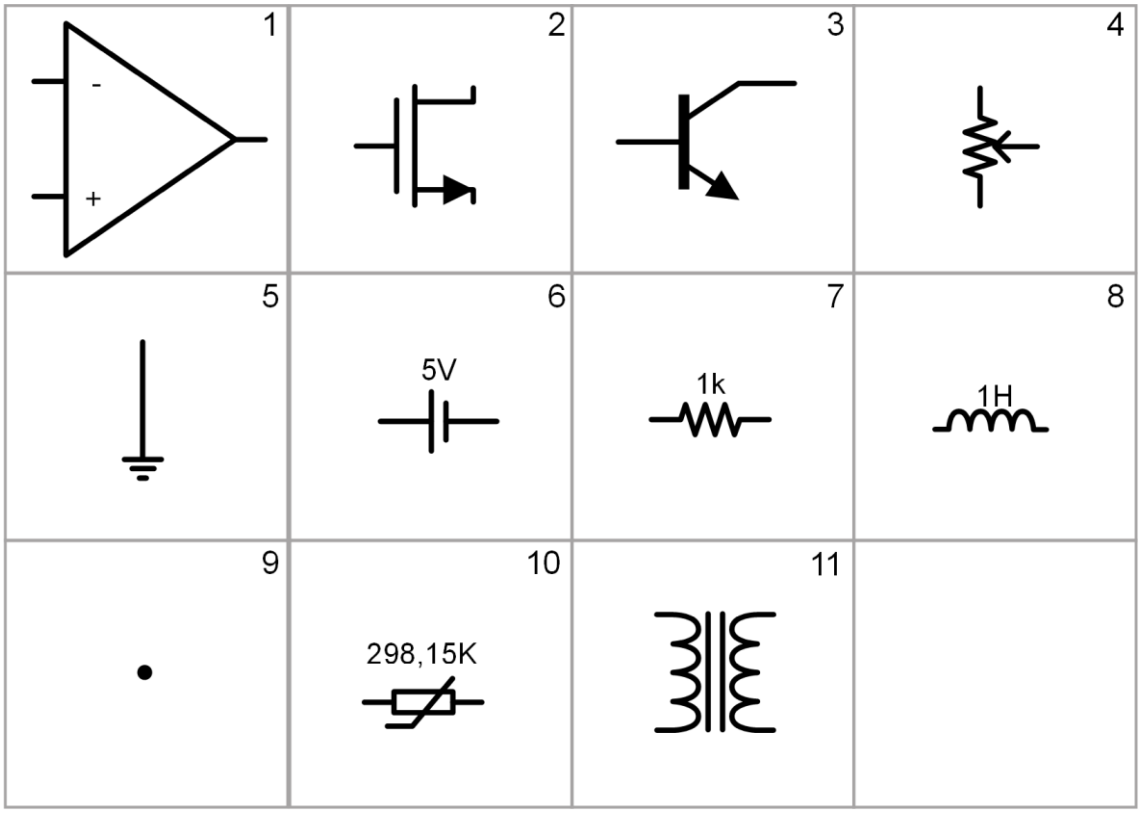


Tabla 4.1: *Lista de elementos que se van analizar.*

1. **Amplificador operacional o A.O**
2. **Mosfet o transistor**
3. **BJT**
4. **Potenciómetro**
5. **Tierra**
6. **Voltaje**
7. **Resistencia**
8. **Inductor**
9. **Nodo**
10. **Termistor**
11. **Transformador**

Aunque algunos elementos tengan valores asociados, pueden variar: el voltaje no es siempre de 5 voltios, sin embargo como siempre va a aparecer en la imagen, se refleja a modo de ejemplo.

4.1.2. Fundamento OCR

La aplicación básicamente consiste en el reconocimiento y posterior tratamiento de los píxeles de una imagen para interpretar y plasmar su significado.

Esta imagen es el resultado final de un determinado software de creación de circuitos. Dicho software exporta las imágenes a una resolución específica para poder ser visualizados por otros programas o aplicaciones, pues no siempre el usuario dispone del software de creación de circuitos.

Los programas de creación de circuitos electrónicos trabajan siempre en formato *vectorial*, lo que significa que no emplean píxeles para su diseño; sólo utilizan píxeles para mostrar el resultado por pantalla.

De esta manera, en los programas que trabajan vectorialmente en su proceso de creación la posibilidad de edición, rectificación, fusión y

muchas más acciones es muy versátil. Por ejemplo, en estos programas se puede usar *zoom in* o *zoom out* de la imagen sin límite.

Un ejemplo de este tipo de softwares son los programas que trabajan con textos –letras-. Cuando se escribe una letra se elige un tamaño de fuente y ésta aparece en pantalla. Si posteriormente el usuario realiza un *zoom in* para aumentar el tamaño de la imagen hasta donde permita el programa, siempre se verá correcta y nítidamente, conservando sus relaciones internas de escritura.

En la *Figura 4.1* se observa como la imagen está al 100% de visualización:

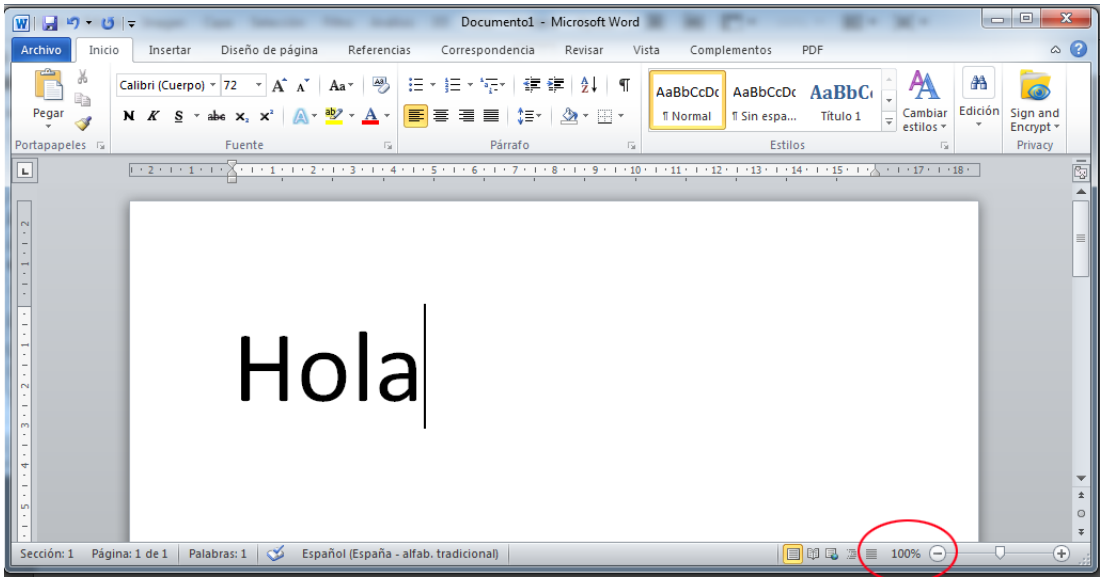


Figura 4.1: *Letras al 100%.*

En la *Figura 4.2*, se aprecia como el *zoom in* ahora está en 270%:

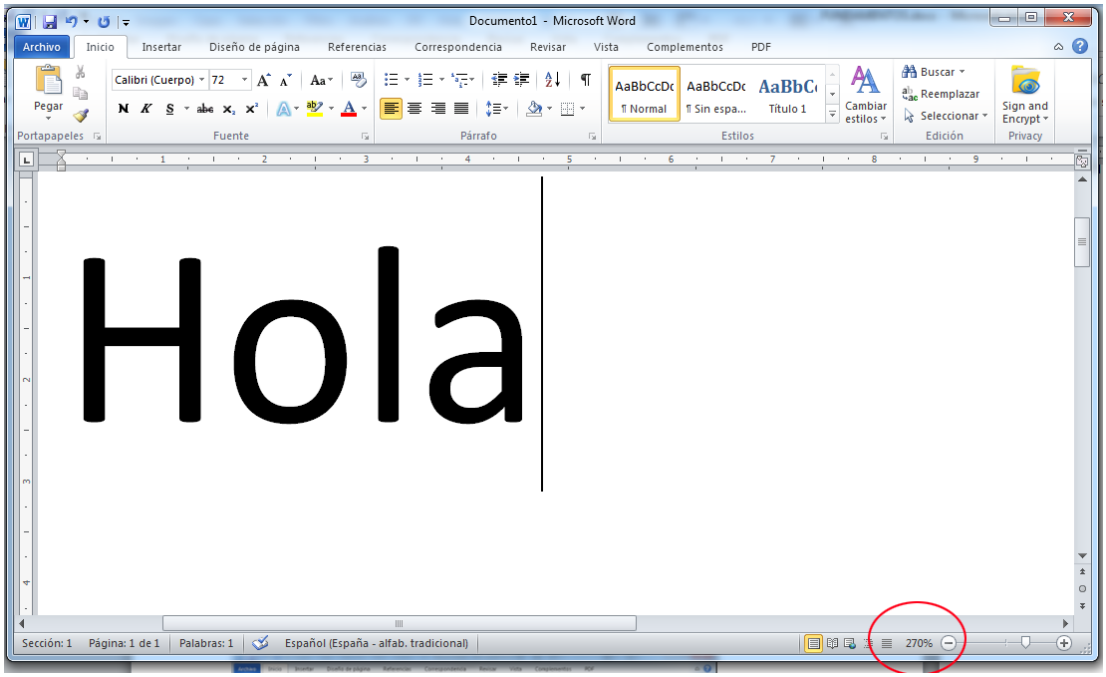


Figura 4.2: *Letras al 270%.*

Si por el contrario, es el tamaño de la fuente la que se aumenta –el tamaño de la letra- ocurre lo mismo, los contornos no pierden calidad. En la *Figura 4.3* se ve como el tamaño de la fuente está en 72:

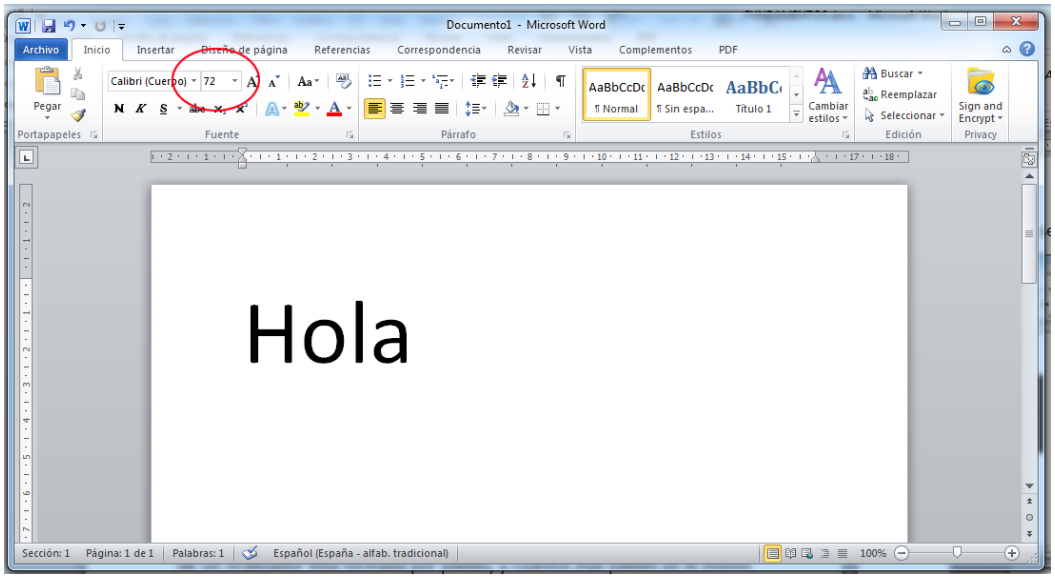


Figura 4.3: *Letras al tamaño de fuente 72.*

En la *Figura 4.4*, el tamaño está en 150, no perdiendo nitidez:

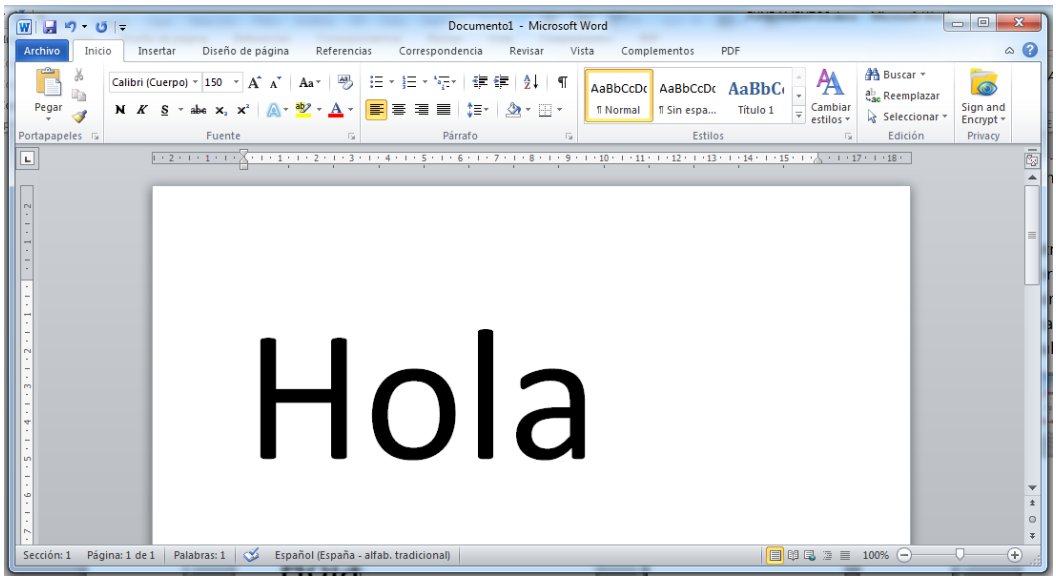


Figura 4.4: *Letras al tamaño de fuente 150.*

Esto se debe a que al ser contornos vectoriales cada vez que se hace una modificación, el programa recalcula la letra y el resultado en pantalla nos muestra unas letras perfectas y a la máxima resolución que permite la pantalla.

Por eso los circuitos que se van a analizar en este proyecto tienen que estar creados por un programa vectorial, de manera que cuando dicho programa exporte las imágenes deban ser de una calidad mínima. Así, estos softwares consiguen que los mismos elementos del circuito tengan siempre las mismas relaciones internas –como el tamaño, etc.- al ser exportados. De forma que cuantos más elementos disponga el circuito, más grande será la imagen para que la imagen no se distorsione.

Si la fuente de exportación de los circuitos no fuera vectorial, el riesgo de error en el análisis es muy elevado.

En la *Figura 4.5*, *Figura 4.6* y *Figura 4.7*, se ve un nodo exportado en dicho programa:

Figura 4.5: *Nodo de un programa vectorial.*

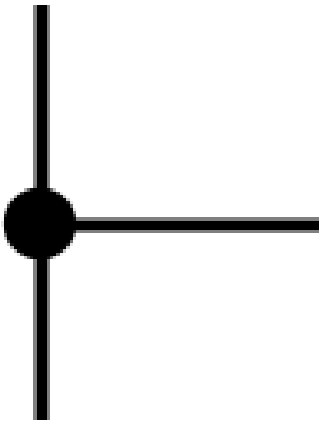


Figura 4.6: *Nodo de un programa vectorial.*

Incluso haciendo *zoom in*, la calidad es perfecta:

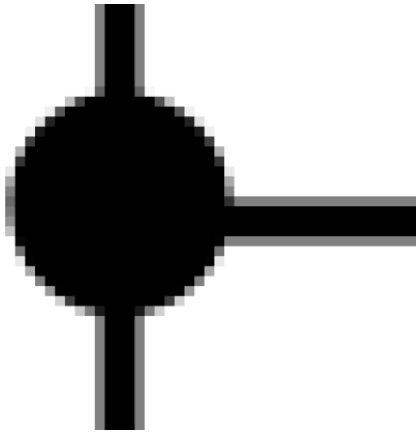


Figura 4.7: *Zoom de un nodo de un programa vectorial.*

Se aprecia que su contorno es nítido y que la cantidad de píxeles grises es muy reducida. Cuando se realicen las operaciones con píxeles pertinentes, éstas serán de calidad.

Sin embargo, si la fuente de exportación de los circuitos fuera un documento escaneado y fotocopiado, la calidad se vería gravemente mermada y su posterior procesado sería muy problemático. Se puede comprobar en la *Figura 4.8*, *Figura 4.9* y *Figura 4.10*:

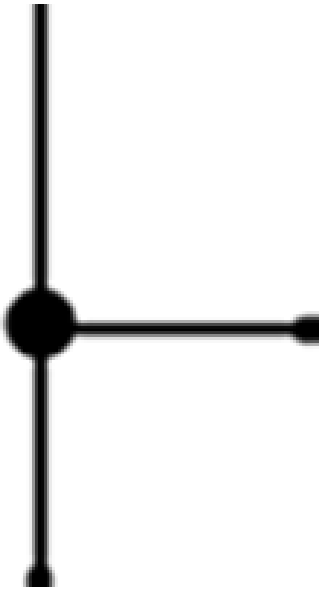


Figura 4.8: *Nodo de un programa no vectorial.* **Figura 4.9:** *Nodo de un programa no vectorial.*

A simple vista parece de la misma calidad que las imágenes anteriores, pero realizando un *zoom in*, se aprecia la diferencia –los contornos aparecen muy difuminados, con una gran cantidad de píxeles grises-:

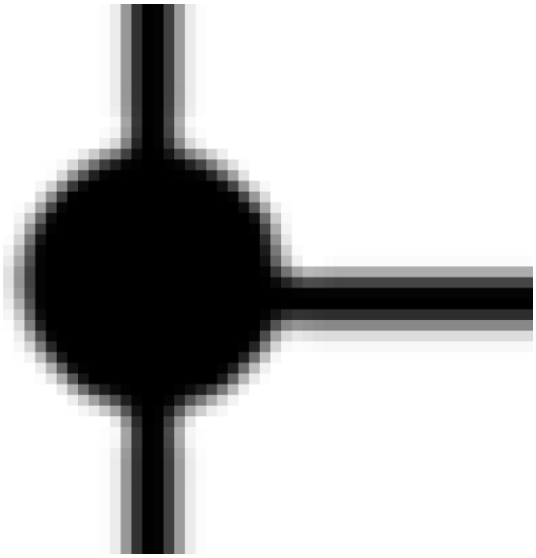


Figura 4.10: *Zoom de un nodo de un programa no vectorial.*

En resumen, sólo serán aptos para la extracción de información aquellos diagramas cuyos elementos sean homogéneos con el patrón que se haya definido para su muestreo, es decir, con un mínimo de calidad.

4.2. Código del programa

Llegados a este punto, se explicará detalladamente cada paso del código del programa; aunque antes de eso se expondrá un flujograma de todos los pasos para que sea más cómodo seguir la explicación.

4.2.1. Flujograma

El principal objetivo del programa es, dada una imagen de un circuito eléctrico, devolver por pantalla la información relevante de éste, a saber: cuántos elementos hay, qué es cada elemento y cómo están conectados entre ellos.

Para llegar a ese objetivo, el programa se ha dividido en tres trozos más pequeños de código, cada uno con su objetivo específico.; éstos se ejecutan de manera consecutiva, alcanzando el objetivo principal del programa. Los trozos de códigos son los siguientes: ***Leer imagen***, ***Identificar elementos del circuito*** y ***Conexión del circuito***.

Primero se explicarán a grandes rasgos los objetivos específicos de estos subprogramas, definiendo claramente lo que hace cada uno. Después se entrará en detalle con cada subprograma, indicando qué pasos lo componen y cómo y con qué fin se ejecutan.

Se va a mostrar de manera gráfica cómo se divide el programa, primero en los tres subprogramas y luego, cómo se divide cada subprograma (*Tabla 4.2*).

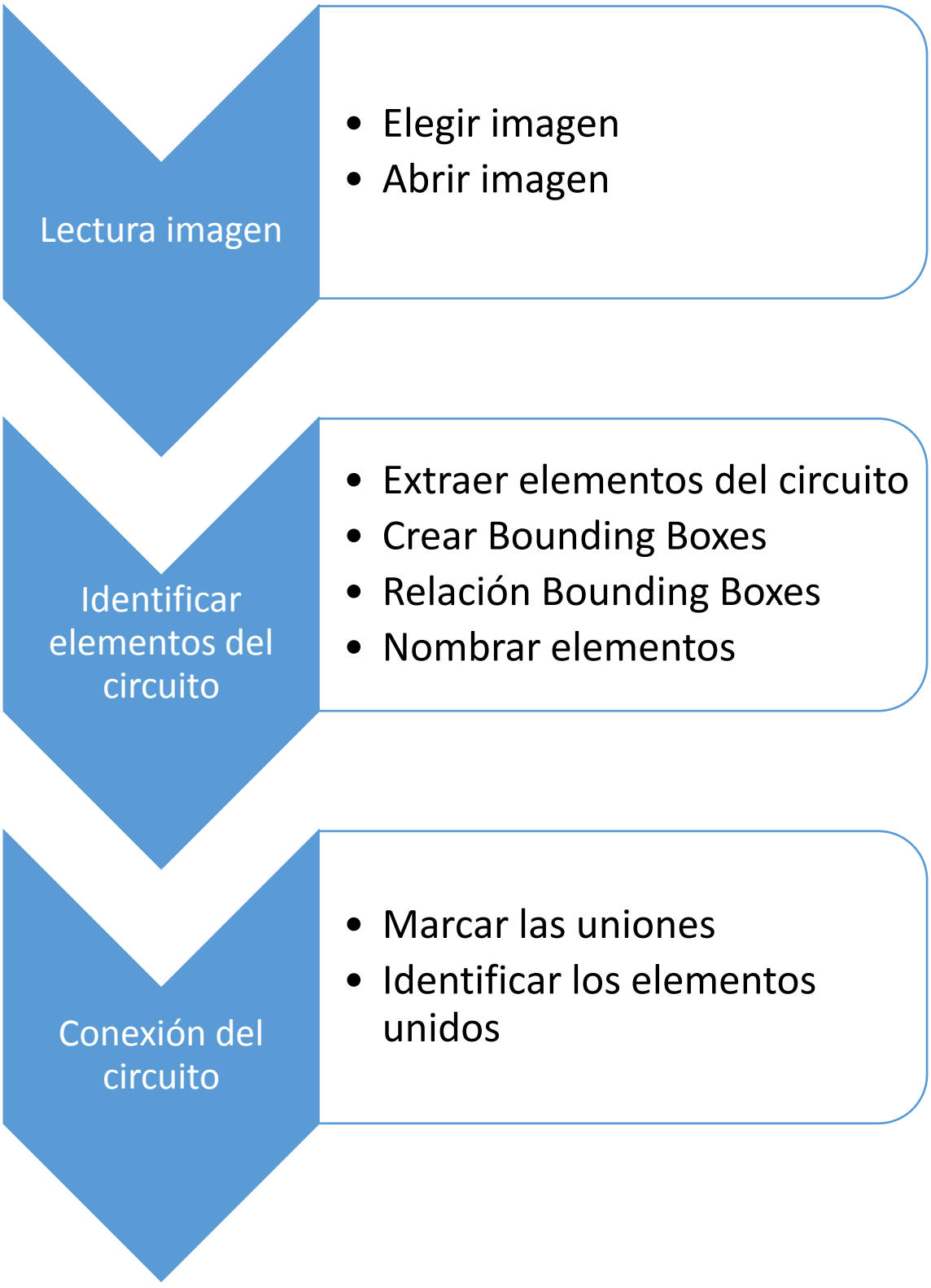


Tabla 4.2: *Flujograma del programa.*

4.2.2. Desarrollo de los subprogramas

4.2.2.1. Lectura imagen

Ésta es la primera parte del programa, donde se elegirá la imagen sobre la que se quiera extraer información. Una vez seleccionada es utilizada por el siguiente subprograma: ***Identificar elementos del circuito***.

1. Elegir imagen

La elección de la imagen se hará de una manera fácil e intuitiva. La interfaz está preparada para seleccionar la imagen directamente desde el fichero que se elija.

2. Abrir imagen

Una vez seleccionada la imagen sobre la que se extraerá información, el programa accederá a la ruta de memoria donde esté guardada la imagen.

La apertura de la imagen se ha realizado mediante un *switch* para que el programa se cierre en el caso de que no pueda ser cargada la imagen.

La imagen seleccionada se guarda en una variable de tipo *Mat* llamada *img_original_0*.

4.2.2.2 Identificar elementos del circuito

Esta parte constituye el segundo subprograma del programa, y es la parte más larga y compleja. Consiste en poder discernir cuántos elementos tiene el circuito de la imagen y qué es cada elemento. No es tarea sencilla, ya que primero se debe eliminar el cable que une todos los elementos (para tenerlos aislados) y, una vez se tenga todos los elementos aislados, se debe poder deducir qué son.

1. Extraer elementos del circuito

El primer paso para poder extraer de forma aislada los elementos es quitar el cable que los une. Al hacer esto, queda en la imagen varios elementos aislados que son más fáciles de tratar que todo el circuito como una figura conjunta; así, llegamos a tener tantas figuras aisladas como elementos hay en el circuito. La lógica usada se muestra en la *Figura 4.11*:

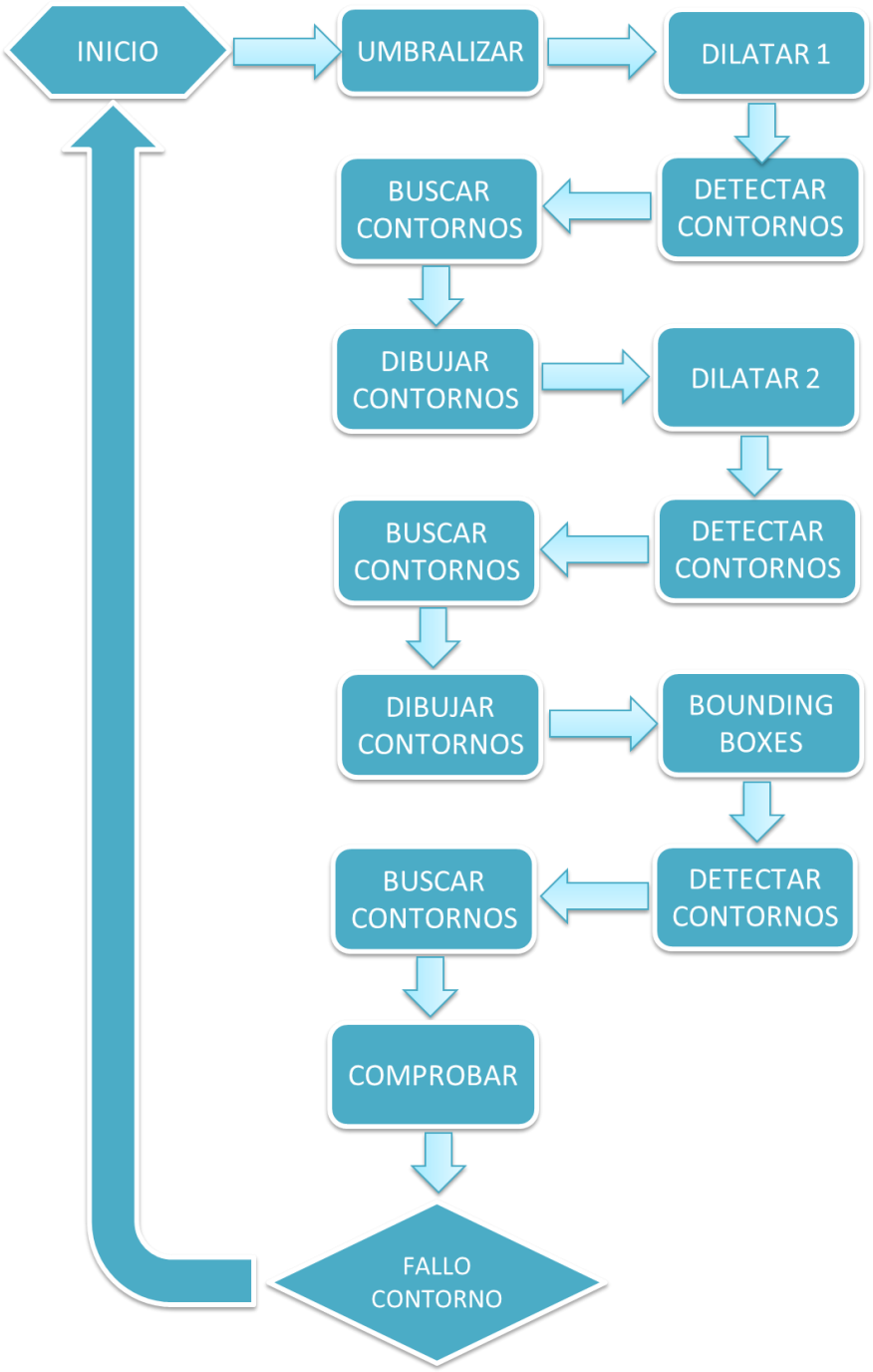


Figura 4.11: *Lógica para extraer los elementos aislados del circuito.*

Todo el proceso de la Figura 4.11 está dentro de un *do while,* que es un ciclo condicional. De esta forma, el programa no sigue adelante a menos que se cumplan los dos requisitos últimos: *fallo* debe ser uno y *contornos.size( )* mayor que cuatro. El objetivo de este fragmento de código es aislar los elementos del circuito.

El ciclo comienza umbralizando la imagen para dejarla en blanco y negro, ya que así es más fácil de tratar (se eliminan un montón de píxeles innecesarios para el proceso). Después, elimina una capa de píxeles y cada vez que se ejecuta va quitando una nueva capa, por eso se usa un contador *dilatar* que va aumentando con cada vuelta (*Figura 4.12*).

```
//DILATAR (para eliminar el cable y quedarme con los elementos aislados)
dilate(img_original_0, img_dilatada, Mat(), Point(-1, -1), dilatar, 1, 1);
```

Figura 4.12: *La función dilate con el contador.*

Sobre la imagen ya dilatada se aplica un detector de bordes *Canny*, que se encarga de extraer los bordes de la imagen a partir de sus variaciones de gradiente (cambios bruscos de intensidad del color). Así, extraería los bordes de todos los elementos. La imagen obtenida quedaría así (*Figura 4.13*):

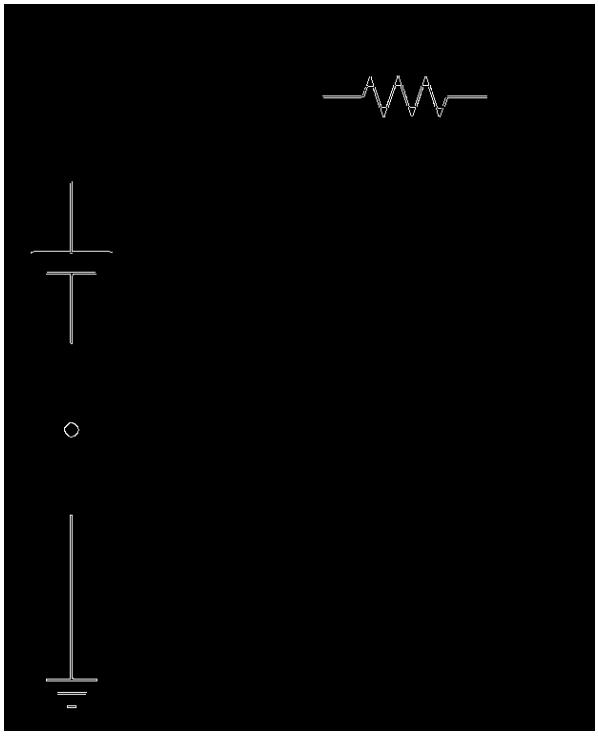


Figura 4.13: *Elementos tras usar Canny.*

La función *findContours* busca los bordes en la imagen, es decir, los detectados por *Canny*, guardándolos como vectores de puntos, siendo cda vector un contorno.

Una vez se han guardado los contornos, se dibujan sobre una nueva imagen para tener una forma sobre la que poder trabajar; se van a dibujar en verde (espacio de color RGB). La imagen obtenida es la siguiente (*Figura 4.14*):

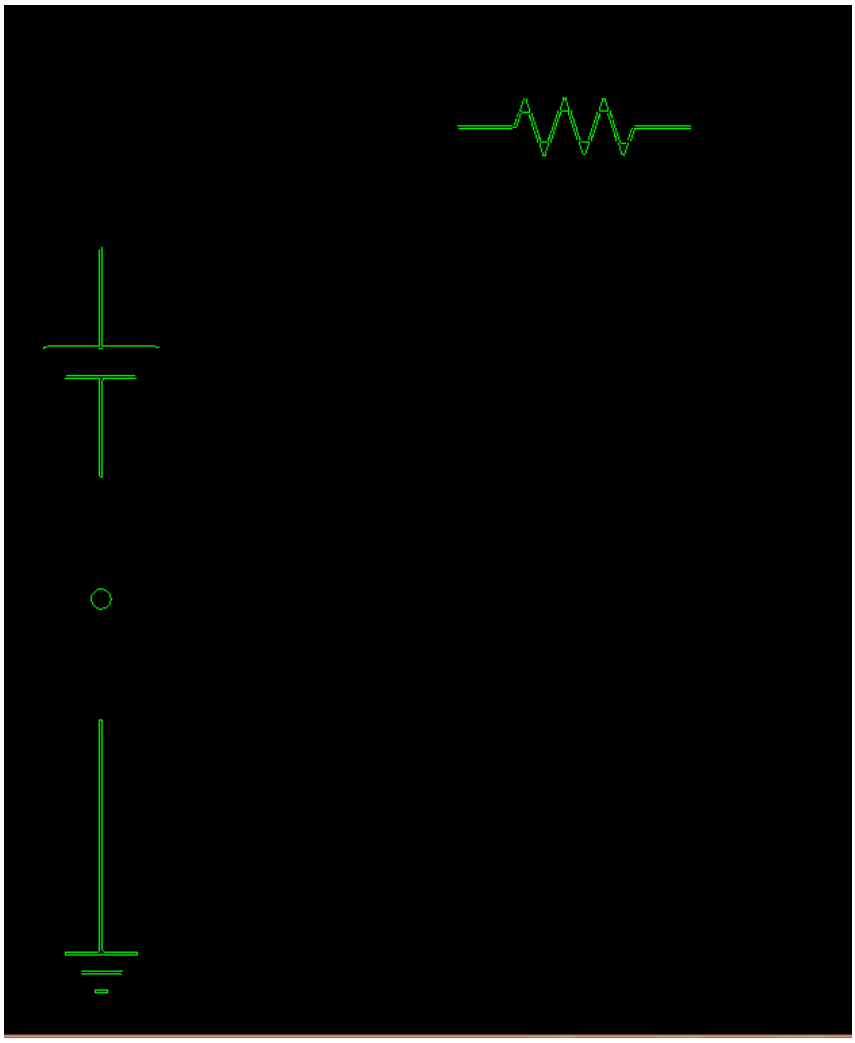


Figura 4.14: *Elementos tras pintarlos de verde.*

Una vez obtenida esa imagen, es fácil percatarse de que cada elemento está formado por varias figuras o contornos. Por ello, los siguientes pasos tienen como objetivo que cada elemento forme una sola figura (un solo contorno) para poder trabajar de una manera más eficiente. Básicamente, los siguientes pasos son iguales que los realizados hasta ahora, pero esta vez la función *dilate* expande las figuras dibujadas en verde, con el objetivo de unir los contornos de cada elemento en uno solo (debido a eso, dilato 12 veces).

La imagen obtenida es la *Figura 4.15*:

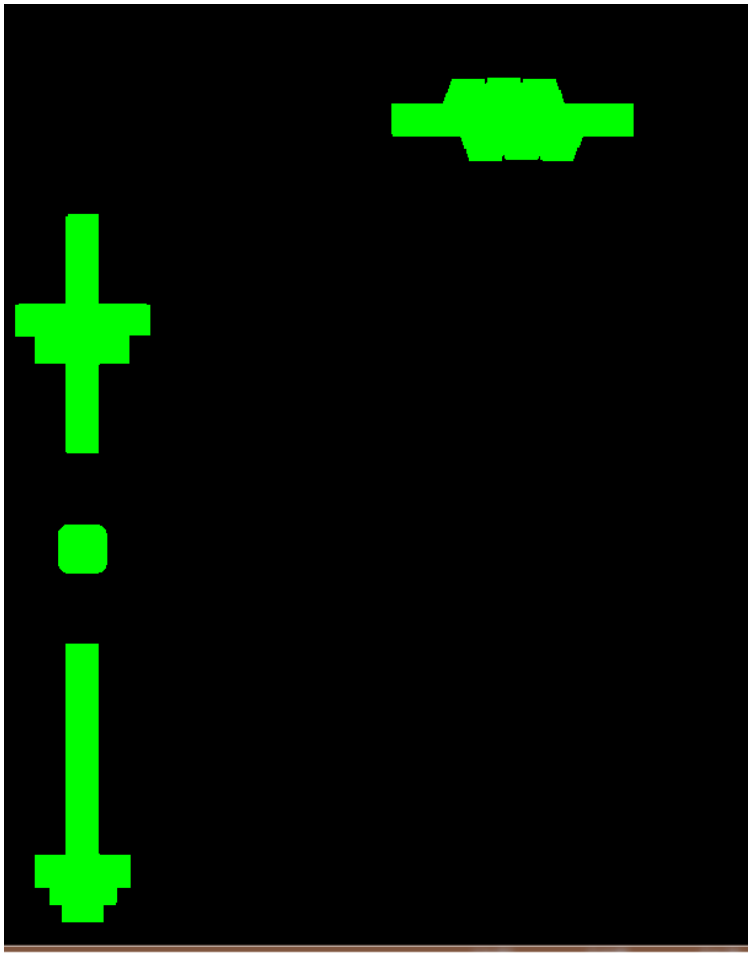


Figura 4.15: *Elementos tras dilatar 12 veces.*

El siguiente paso es usar una serie de funciones (*Bounding boxes*) con el objetivo de "encerrar" dichas figuras en rectángulos, cuyas coordenadas (una serie de puntos de tipo *Point*) son las que se usarán para las siguientes operaciones. Una vez realizado esto, se vuelve a usar la función *Canny* para encontrar los límites y luego un *findContours* para obtener los puntos de esos rectángulos.

Para finalizar este bucle, indicar cuales son las condiciones de salida. Es necesario que los contornos encontrados sean más de cuatro: para el funcionamiento mínimo de un circuito eléctrico es necesario contar con cuatro elementos básicos – a saber, **voltaje**, **resistencia, tierra** y **nodo** -, por lo que mientras el *do while* encuentre menos de cuatro elementos, considera que aún no ha eliminado el cable del circuito. La segunda condición para salir del bucle es que no haya ningún contorno que exceda un tamaño máximo (este valor viene marcado por el mayor elemento que puede encontrar).

Si se cumplen ambas condiciones, considera que ya ha extraído los elementos y termina el bucle.

2. Crear Bounding Boxes

En posesión de las coordenadas de los rectángulos que encierran los elementos del circuito se realizan los siguientes pasos.

Lo primero es hacer una copia idéntica de la imagen original mediante la propiedad de la clase *Mat* llamada *clone( )*. Se cambia la imagen de espacio de color, de escala de grises (GRAY) a formato BGR (Blue, Green, Red) con la función *cvtColor*.

Con las coordenadas, se crean propiamente dicho los rectángulos con la función *boundRect*, ya que se necesitarán para pintar en la imagen original del circuito los rectángulos sobre cada elemento.

La idea de dejar los píxeles del cable en un negro y los rectángulos en un verde perfectos se explicará más adelante, cuando sea necesario interpretar la conectividad del circuito.

La *Figura 4.16* muestra la salida de este proceso:

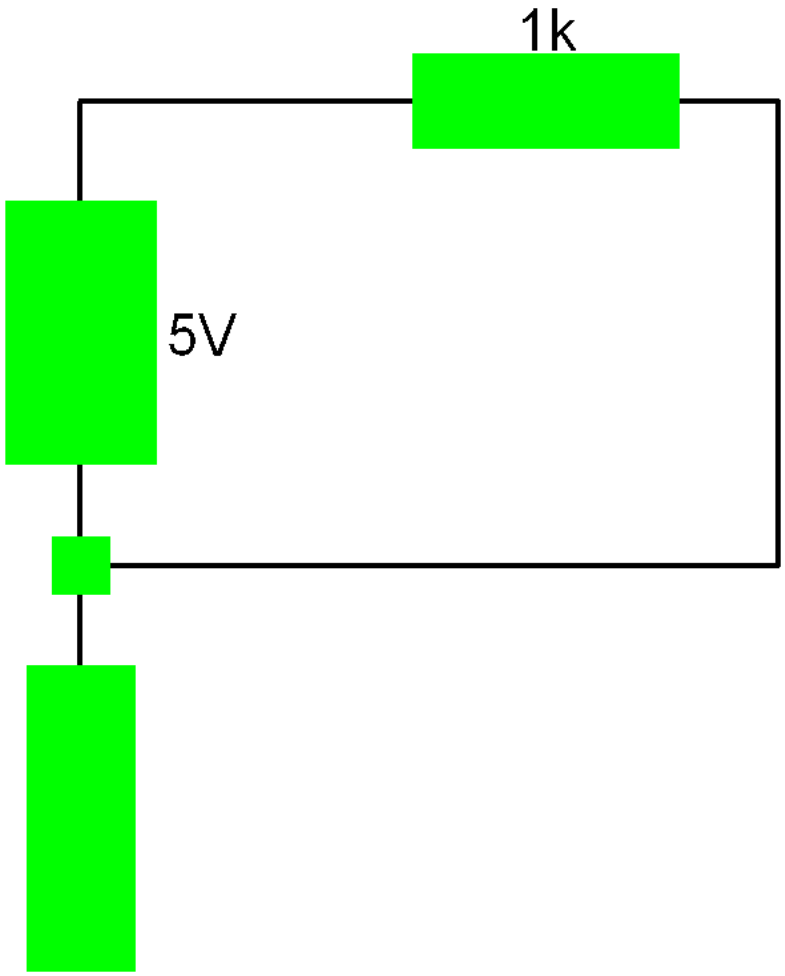


Figura 4.16: *Crear Bounding boxes.*

3. Relación Bounding Boxes

En este punto ya se comienza a extraer alguna información relevante para la comprensión del circuito. Todos los elementos se encuentran ahora encerrados en unos rectángulos que serán iguales –si no iguales, muy similares; todo conserva siempre algo de error- en todos los circuitos (ya que todas las imágenes guardan una relación de aspecto o resolución), por lo que las dimensiones de estas cajas serán conservadas como si de un patrón se tratase. En función de la interpretación de estos valores, se pueden sacar conclusiones sobre los elementos de la imagen; una de ellas

es que la relación *alto – ancho* de las cajas se va a mantener constante, indiferentemente de si el elemento estuviera girado o no.

Por ello se hace necesario extraer estas relaciones de la imagen, y guardarlas en un vector que se usará más adelante, cuando hay que asignarle el nombre los elementos.

Es importante que se mantenga cierta disciplina a la hora de extraer la relación, y por ello siempre será el lado mayor entre el lado menor.
A modo de ejemplo, se mostrará por pantalla cómo varían las relaciones según el elemento que se observe (*Figura 4.17*):

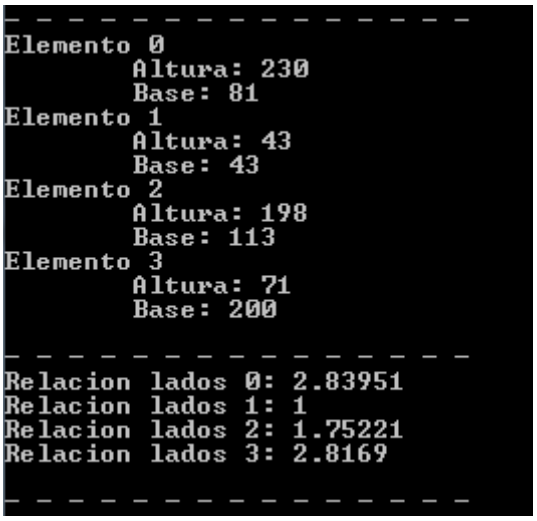


Figura 4.17: *Relación lados elementos.*

Se puede observar, por ejemplo, que la relación de lados del elemento 0 es muy similar a la del elemento 3. No pasaría nada, ya que si se observan sus alturas y sus bases, hay una gran diferencia entre los valores más elevados -230 y 200-; luego esa sería una segunda condición para distinguir entre ambos elementos.

4. Nombrar elementos

El objetivo de este proceso es mostrar por pantalla la imagen inicial del circuito con el nombre de los elementos escritos en él. Los nombres aparecerán en la parte superior de los elementos, con un número delante indicando una cierta guía para orientarse. Además, la pantalla de la aplicación muestra cuántos elementos ha encontrado en la imagen así como el nombre de todos los elementos, numerados en una lista. Es una manera mucho más rápida de comprobar los elementos que hay y qué número tienen asignados.

La numeración que es dada a los elementos responde al orden en que los encuentra el programa. Por ello, si se mira la imagen de salida con los nombres escritos, no sigue un orden aparente. También se puede comprobar que algunos números no aparecen en la lista, como si se los saltase; nada más lejos de la realidad: los números que no aparecen están asociados al ruido[1] de la imagen, por lo que cuando el programa tiene que detectar los elementos, el ruido, al no cumplir con los parámetros que los definen, se ignora.

La importancia de la numeración de los elementos no es simplemente por comodidad de comprensión o por cuidado del formato, es muy necesaria a la hora de comprobar la conexión del circuito. Es como una guía que usará el programa para ver cómo están conectados los elementos. Esta parte se explicará en detalle más adelante, en ***Conexión del circuito***.

Varios ejemplos de cómo quedaría la imagen que mostrará el programa son los siguientes:

[1] El ruido de una imagen es cualquier píxel o conjunto de píxeles que, o bien son ajenos a la imagen (puede ser una distorsión de la imagen original por diferentes efectos, como la luz), o bien no son necesarios para el objetivo sobre el que se esté trabajando.

- Un circuito sencillo que sólo consta de un voltaje, una resistencia, un nodo y una tierra (*Figura 4.18*).

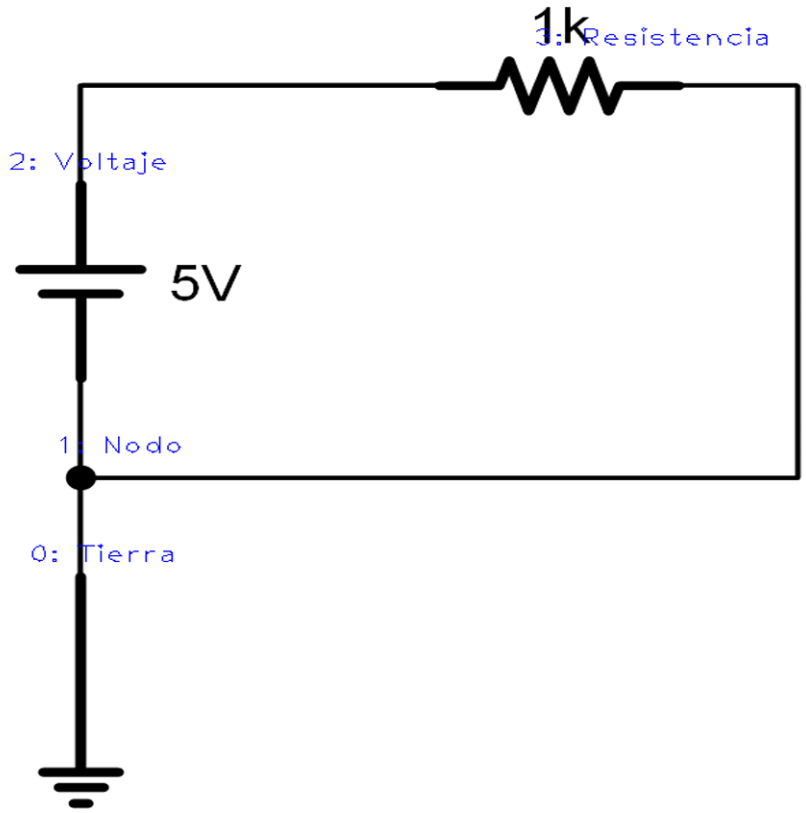


Figura 4.18: *Circuito de ejemplo 1.*

La forma de anotación es la siguiente:

0. Tierra
1. Nodo
2. Voltaje
3. Resistencia

Por la pantalla de la aplicación se mostraría la *Figura 4.19*:

```
- - - - - - - - - - - - - - - -
Lista de elementos de la imagen

0:Tierra
1: Nodo
2:Voltaje
3:Resistencia
---------------------------------------
El numero total de elementos es: 4
---------------------------------------
```

Figura 4.19: *Relación lados elementos ejemplo 1.*

Se ve como aparece primero la lista numerada de los elementos y después el número total de elementos en la imagen.

- Un circuito algo más complejo, que consta de dos voltajes, tres resistencias, un potenciómetro, un mosfet, un amplificador operacional (A.O, por abreviatura), dos tierras y un nodo (*Figura 4.20*).

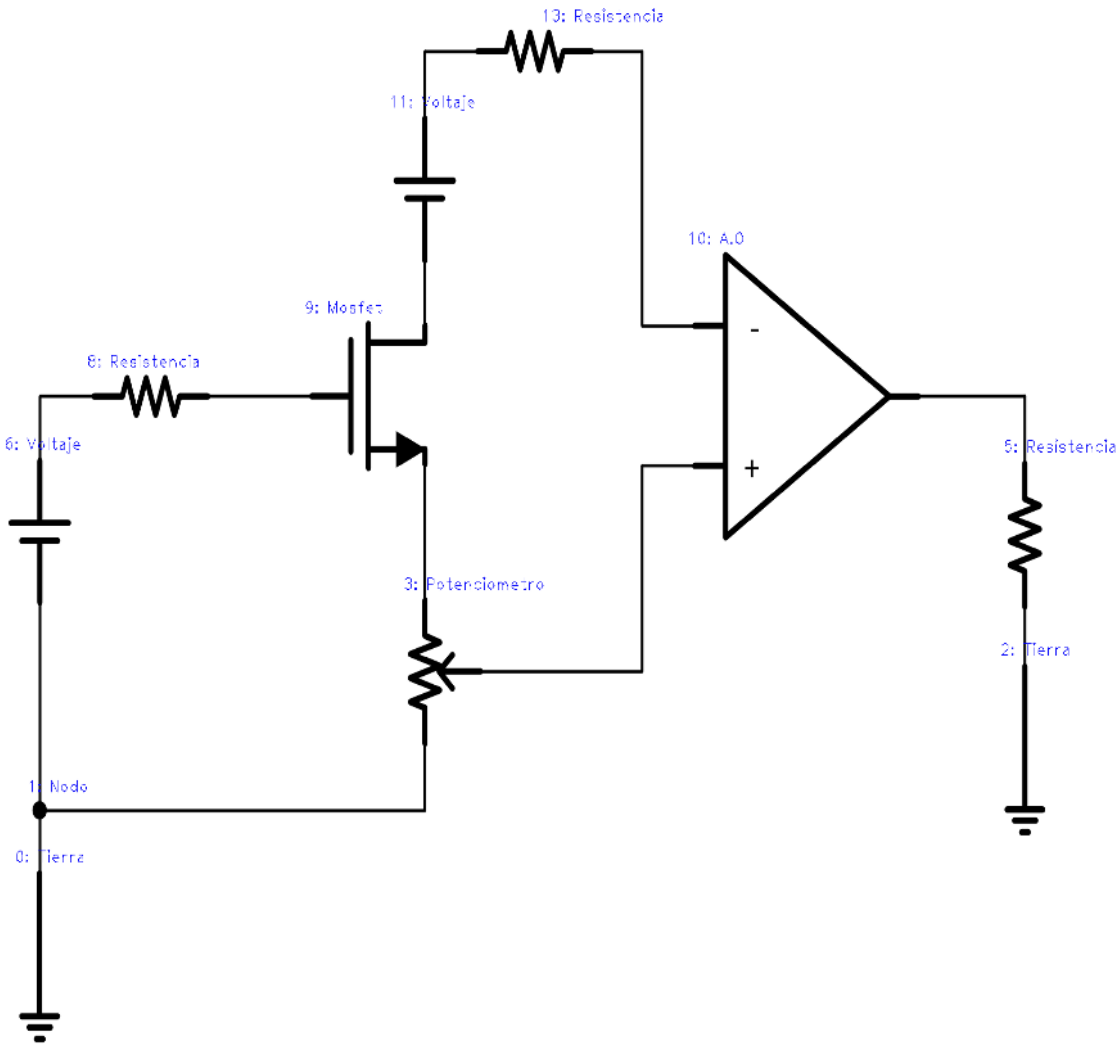


Figura 4.20: *Circuito de ejemplo 2*

.

La forma de anotación es la siguiente:

0. Tierra

1. Nodo

3. Potenciómetro

5. Resistencia

6. Voltaje

8. Resistencia

9. Mosfet

10. A.O. (Amplificador operacional)

11. Voltaje

13. Resistencia

Por la pantalla de la aplicación se mostraría la *Figura 4.21*:

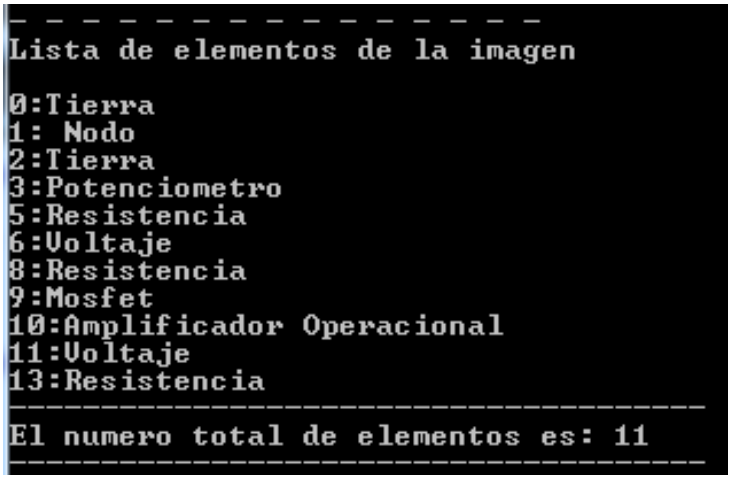


Figura 4.21: *Relación lados elementos ejemplo 2.*

- Este circuito consta de un voltaje, cuatro resistencias, cuatro condensadores, cinco nodos y una tierra (*Figura 4.22*).

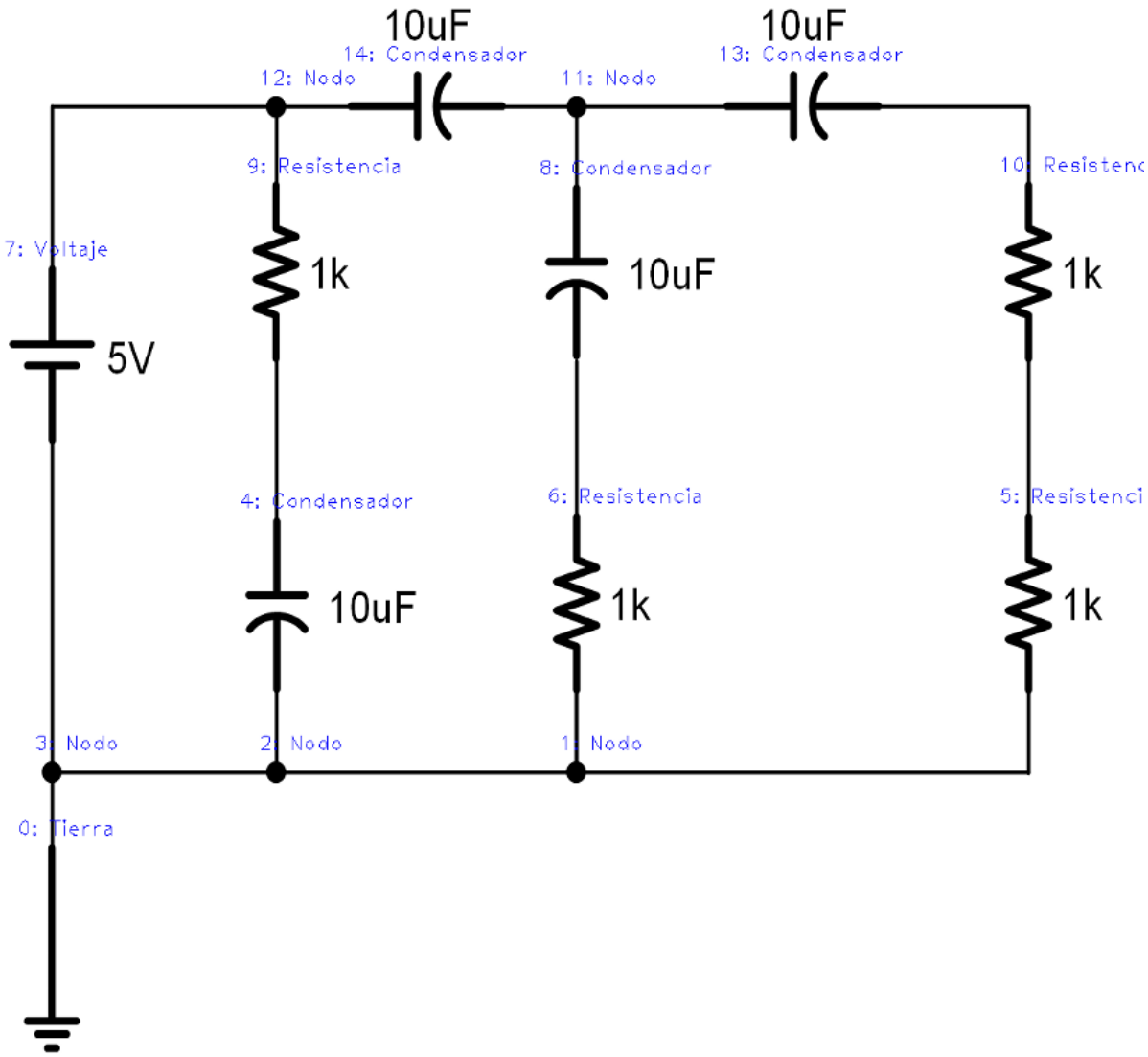


Figura 4.22: *Circuito de ejemplo 3.*

La forma de anotación es la siguiente:

0. Tierra
1. Nodo
2. Nodo
3. Nodo
4. Condensador
5. Resistencia
6. Resistencia
7. Voltaje
8. Condensador
9. Resistencia
10. Resistencia
11. Nodo
12. Nodo
13. Condensador
14. Condensador

Por la pantalla de la aplicación se mostraría la *Figura 4.23*:

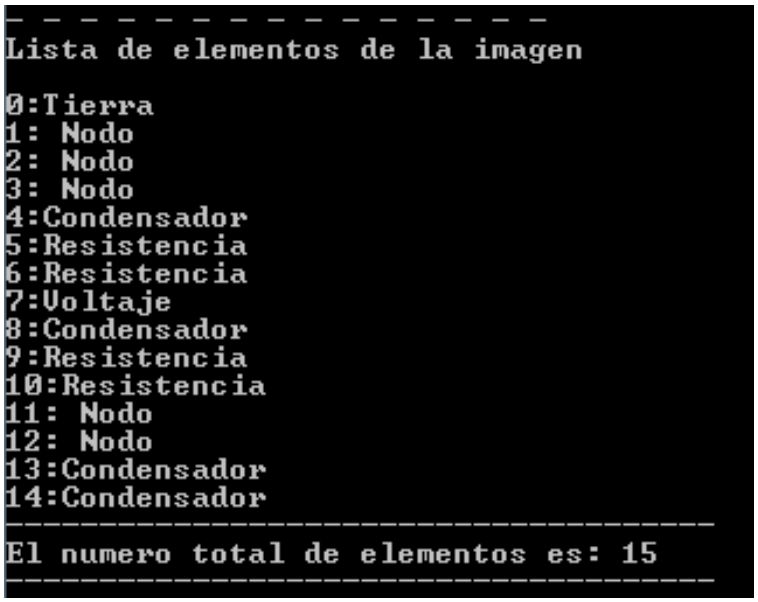


Figura 4.23: *Relación lados elementos ejemplo 3.*

- Este circuito consta de tres tierras, cinco nodos, un voltaje, seis resistencias, un amplificador operacional y un mosfet (*Figura 4.24*).

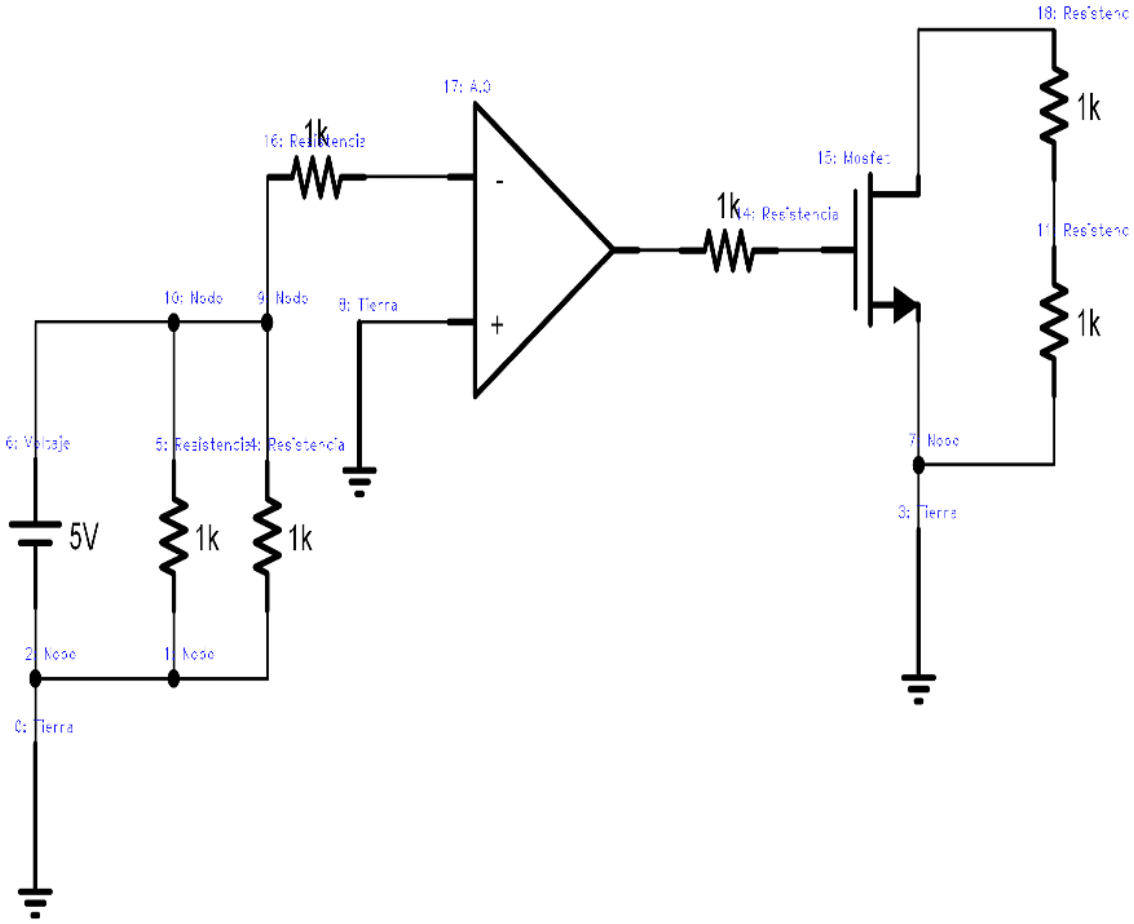


Figura 4.24: *Circuito de ejemplo 4.*

La forma de anotación es la siguiente:

0. Tierra
1. Nodo
2. Nodo
3. Tierra
4. Resistencia
5. Resistencia

6. Voltaje
7. Nodo
8. Tierra
9. Nodo
10. Nodo
11. Resistencia
14. Resistencia
15. Mosfet
16. Resistencia
17. A.O. (Amplificador operacional)
18. Resistencia

Por la pantalla de la aplicación se mostraría la *Figura 4.25*:

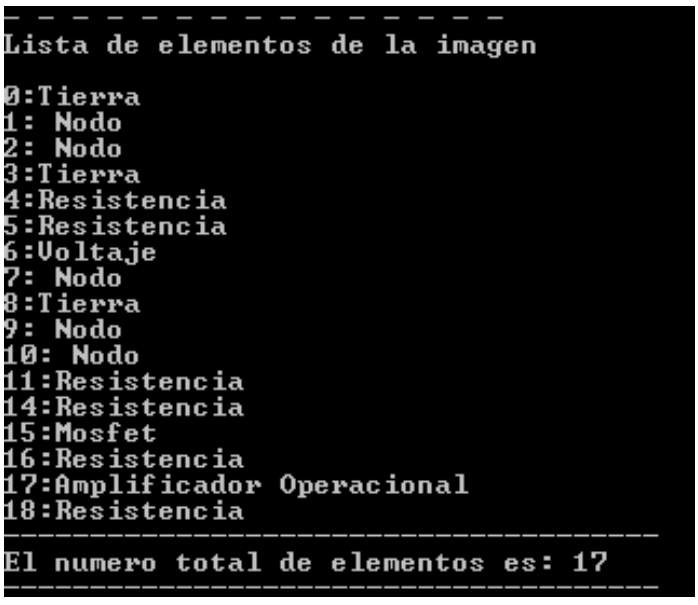


Figura 4.25: *Relación lados elementos ejemplo 4.*

- Este circuito contiene dos tierras, un voltaje, un BJT, un mosfet, dos nodos y tres resistencias (*Figura 4.26*).

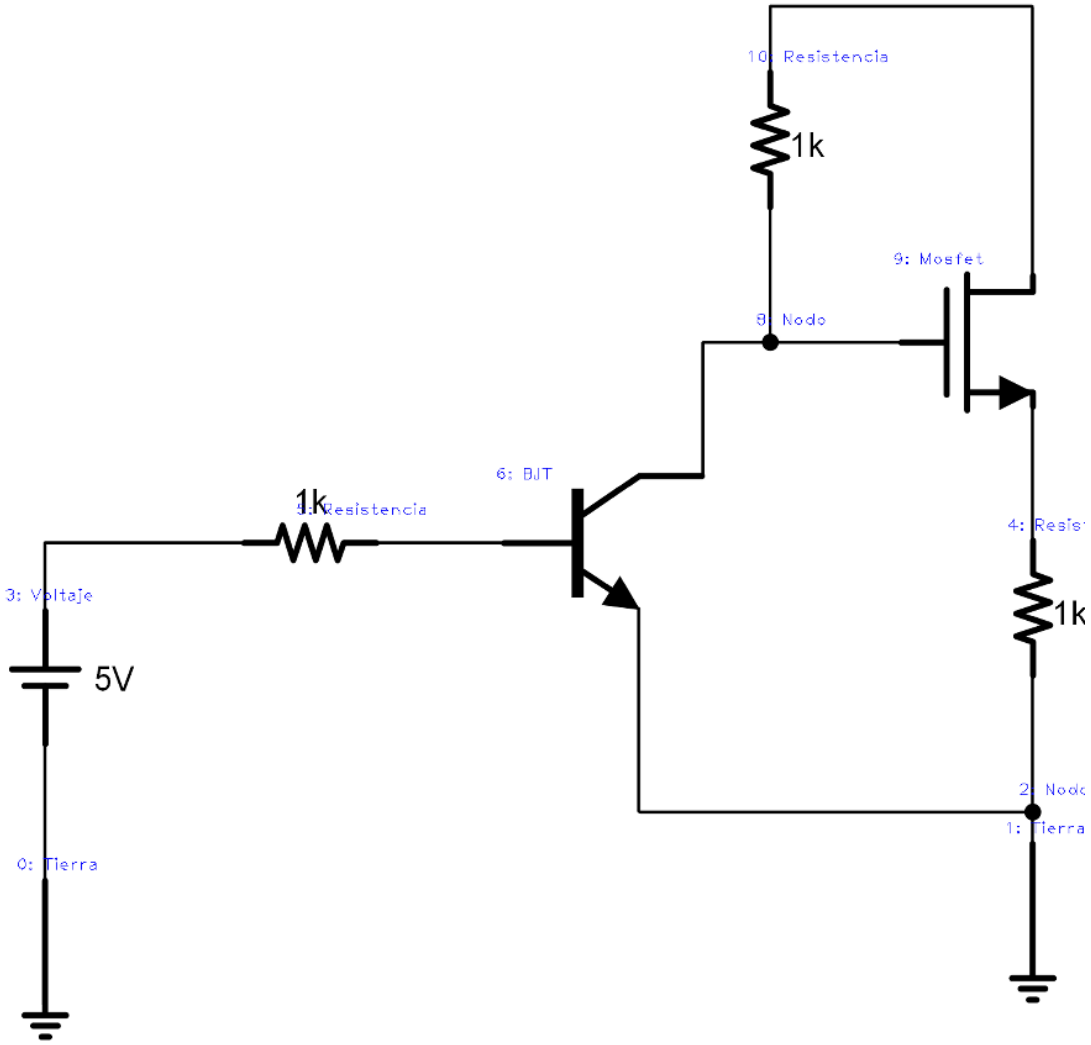


Figura 4.26: *Circuito de ejemplo 5.*

La forma de anotación es la siguiente:

0. Tierra

1. Tierra

2. Nodo

3. Voltaje

4. Resistencia

5. Resistencia

6. BJT

8. Nodo

9. Mosfet

10. Resistencia

Por la pantalla de la aplicación se mostraría la *Figura 4.27*:

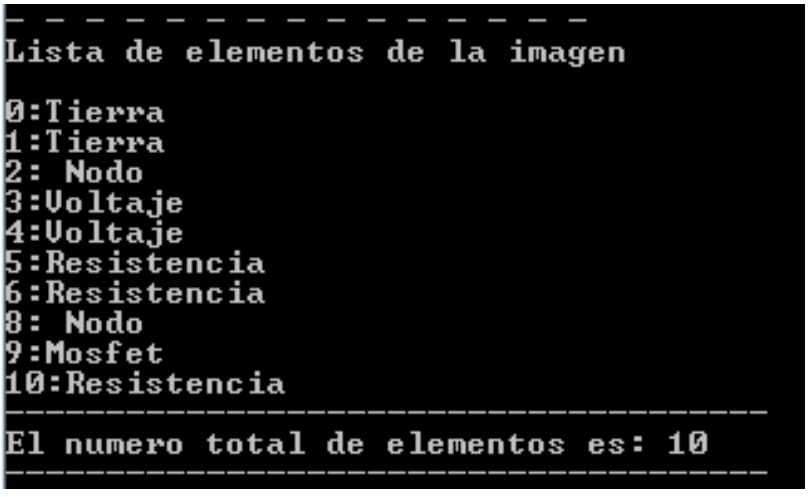


Figura 4.27: *Relación lados elementos ejemplo 5.*

Para decidir qué es cada elemento, se ha usado una cascada de sentencias *if*, una detrás de otra, con todas las condiciones necesarias para establecer la pertenencia a un elemento concreto.

A partir de todas las relaciones de los lados de los rectángulos obtenidos en la anterior parte del programa –***Relación Bounding Boxes-***, se establecen unos rangos de pertenencia para cada elemento. Estos rangos se han establecido obteniendo valores de las relaciones para cien circuitos distintos; con todos esos valores, se han creado unos valores máximos y mínimos para cada rango. Cuando el programa analiza una relación de lados de un elemento desconocido, comprueba mediante una sentencia *if* si ese valor está o no dentro del rango de cada tipo de elemento: si está dentro, ese elemento desconocido pertenece al elemento evaluado. Por ejemplo, el rango de valores para el elemento **Potenciómetro** es el siguiente: tiene un **valor máximo** de **1.47** y un **valor mínimo** de **1.42**. Estos valores, para evitar posibles pequeños errores en la medición de los valores, han tenido en cuenta un error al alza (en el caso del valor máximo) y un error a la baja (en el caso del valor mínimo) aumentando dichos valores en un 5%; lo justo para dejar un margen para algún fallo en la obtención de las relaciones.

Si la primera condición fallara, debido a que dos o más elementos comparten rango de valores (como es el caso del **DC Rail** y del **Diodo**), entonces se establece una segunda condición: se mirarán también, por ejemplo, sus longitudes máximas.

Una vez cumplidas estas condiciones, muestra por pantalla la posición donde ha sido encontrado, el nombre del elemento, escribe en la imagen original (*img_original_0*) el nombre del elemento con su número, y suma uno al contador de elementos *numero_elementos* (este contador suma uno cada vez que un elemento desconocido cumple con la condición de pertenencia a algún elemento, por ello está escrito en cada condición *if*).

4.2.2.3. Conexión del circuito

Esta parte añade la última información útil del análisis. Una vez que ya se saben los elementos del circuito, dónde están y qué y cuántos son, lo que queda es saber cómo están conectados.

Para ello, se debe conocer cuántas conexiones de cables tienen los elementos, pues cada una de estas conexiones significará que está unido a otro elemento. Después se deberá recorrer el cable para ver que conexiones están unidas.

De esta forma se puede crear una especie de mapa de conexiones de los elementos, con el objetivo de poder dibujar el circuito a partir de esta información.

1. **Marcar las uniones**

Se parte de la imagen *rect_dibujos*, que es la imagen de salida del proceso *Crear Bounding Boxes*. Ahora es cuando se va a entender por qué dejar esa imagen en dos colores perfectos: verde (RGB: 0, 255, 0) para los elementos y negro (RGB: 0, 0, 0) para el cable. La idea es que el programa sea capaz de detectar qué es cable y qué es elemento, de ahí separarlos por colores.

Por ejemplo, se parte de la *Figura 4.28*:

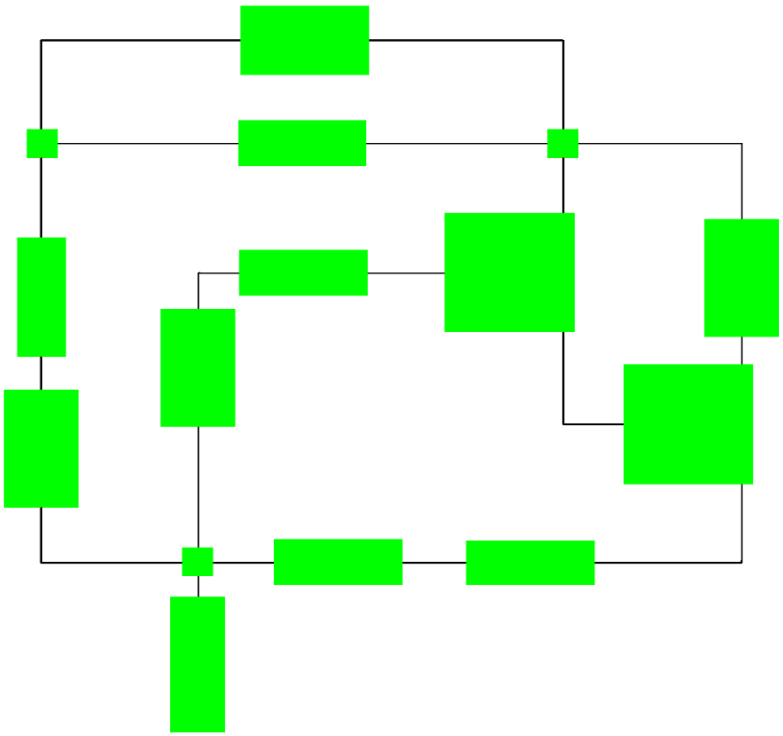


Figura 4.28: *Imagen de partida.*

Después del proceso completo, así quedaría la imagen: elementos en verde, cables en negro, y todas las conexiones del cable al elemento en rojo (RGB: 255, 0, 0). Nótese que los cables quedan de dos, tres o cuatro píxeles de grosor (*Figura 4.29*).

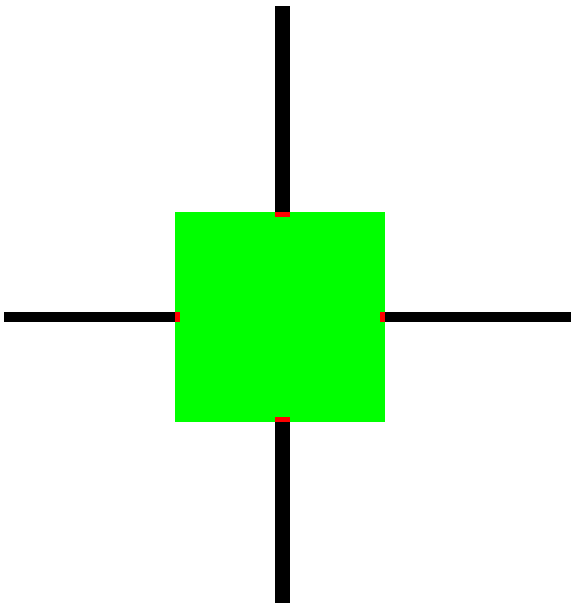


Figura 4.29: *Marcar uniones.*

A pesar de la calidad de la imagen, se aprecian los diferentes grosores de los cables, en el que el inicio y el fin del mismo están delimitados por los píxeles rojos.

2. Identificar los elementos unidos

Ahora se analizan las conexiones, siguiendo esta lógica: como no se sabe el grosor exacto del cable, pero se sabe que puede abarcar desde dos a cuatro píxeles, se deben mirar todos los píxeles desde cada posición en un radio de cuatro píxeles. Se recorre la imagen hasta encontrar un píxel rojo, en ese momento se pinta de azul los píxeles vecinos (incluido él mismo) para marcar como leído esa conexión. Desde esa posición, se comprueba el color de los píxeles vecinos (con el radio indicado) para encontrar los

píxeles negros. Donde se encuentren, significa que es el camino que sigue el cable. Se avanza en esa dirección, pero recordando pintar de azul los píxeles que dejas atrás, para que siempre avance la posición. Cuando se encuentra con otro píxel rojo, es el final del recorrido y, por tanto, del cable: ya se tiene una conexión entre elementos. No olvidar pintar de azul el píxel encontrado, para no volver a encontrarlo. Una vez guardada la posición de los píxeles rojos, se comprueba que contorno los encierra para saber a cuál pertenecen.

El resultado es el de la *Figura 4.30*:

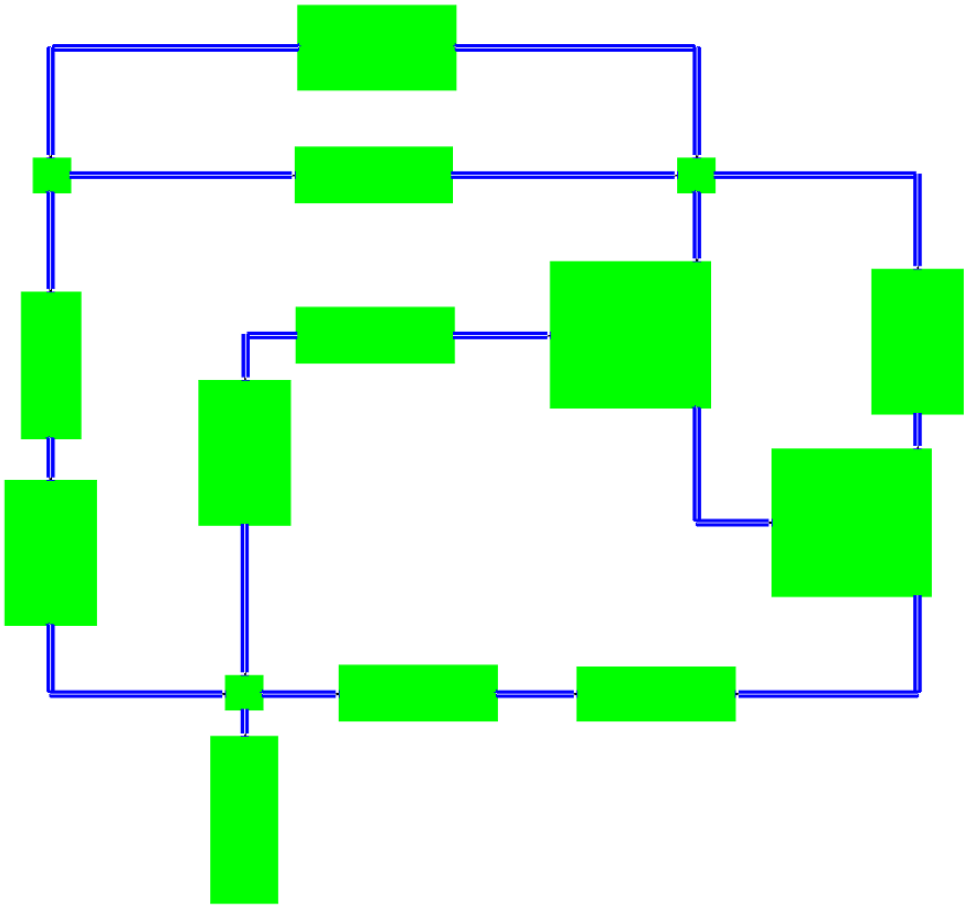


Figura 4.30: *Recorrer cables.*

Y el resultado se muestra por pantalla, indicando qué contornos están unidos (*Figura 4.31*):

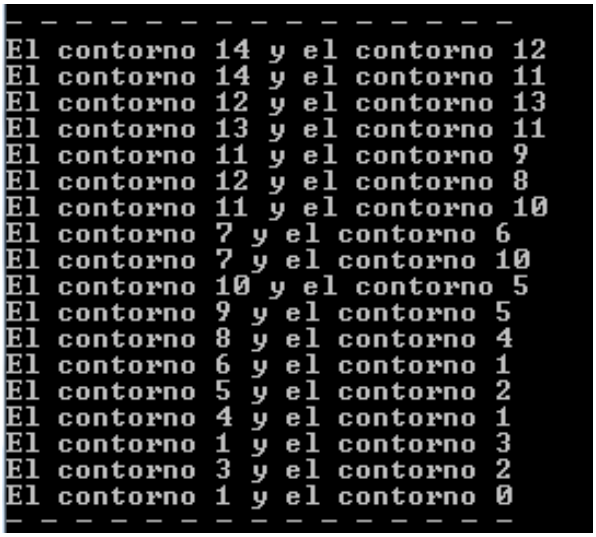


Figura 4.31: *Conexiones por pantalla.*

4.3 Alternativas inviables

Antes de utilizar la exportación de las imágenes bajo el mismo patrón, tal como se explicó en los fundamentos del OCR, se intentó que el programa fuera capaz de analizar las imágenes independientemente de su resolución.

En este apartado aparecerán recogidas de forma breve –sin entrar en detalle- las alternativas que se han intentado para lograr el objetivo del proyecto, y cuyos resultados no han sido favorables. Se recogen aquí para que, si en un futuro alguien intenta implementar alguna idea similar, sepa de antemano qué alternativas no han funcionado.

Todas las alternativas partían de la base de usar una imagen modelo para cada elemento, y usarla para un rápido aprendizaje del programa. Por ello, en una carpeta del programa –del ordenador donde se vaya a utilizar el programa- se introducen todas las imágenes modelo, que luego serán llamadas por el programa para su uso.

Estas son las alternativas de desarrollo que no han funcionado:

- Ajustar tamaño imagen completa: tomando como imagen modelo la del *Nodo*, por ser la más perfecta –relación lados uno, lados más pequeños y, siempre debe aparecer en un circuito-, se comprueba la resolución del *Nodo* en la imagen a analizar y, en consecuencia, se reajusta la resolución de la imagen. Uno de los problemas era el alto coste computacional, que se traduce en un mayor tiempo de ejecución. Otro problema era que no siempre realizaba bien el ajuste, al leer
- mal el tamaño del *Nodo* de la imagen nueva y por lo tanto luego reajustaba mal la imagen, produciendo una muy mala lectura del circuito.

- Ajustar tamaño elementos aislados: muy similar al ajuste de tamaño de imagen completa, pero esta vez a partir de "recortes" de los elementos. Una vez en posesión de los rectángulos de los elementos, sólo hacía falta reajustar dichos recortes –no son más que nuevas imágenes, sólo que más pequeñas-. El coste computacional era más bajo que en otro caso, pero daba el mismo tipo de error: si tomaba mal la dimensión del *Nodo*, arrastraba el error a todos los elementos.

- Correlación de Histogramas: se obtienen los histogramas de las imágenes de los elementos modelo y de los recortes de los elementos nuevos, y se cotejan. En un principio podría haber dado buen resultado, ya que es independiente del tamaño de las imágenes –para comparar dos imágenes es necesario que tengan el mismo tamaño-. Sin embargo, se producían muchos errores ya que cualquier diferencia de píxeles causaba mucha diferencia en los histogramas.

- Restar imágenes: se restan las imágenes de los elementos modelo y las imágenes de los recortes, y se analizan las diferencias. Si el número de píxeles blancos (los píxeles blancos hacen referencia a los píxeles que varían de una imagen a otra; es el resultado de la resta de píxeles entre una imagen y otra) son reducidos, entonces es el mismo elemento. También daba numerosos errores ya que era necesario que las imágenes tuvieran el mismo tamaño, por lo que cualquier mínima variación en los píxeles o cualquier error en el reajuste del tamaño impedía la correcta correlación.

Capítulo 5. Resultados obtenidos

5.1 Descarga de imágenes

El objetivo principal del programa es la extracción de información presente en diagramas de circuitos electrónicos, más concretamente la capacidad de interpretar qué elementos aparecen en él y cómo están conectados.

Durante toda la explicación del código implementado se han utilizado circuitos diseñados por mí en la app iCircuit. Siendo perfectamente válidos para comprobar la eficacia del programa, las pruebas experimentales se han realizado sobre circuitos descargados directamente de la página oficial de iCircuit. De esta forma, se obtienen unos resultados sobre diagramas no diseñados por mí.

Lo primero que se debe hacer, es acceder al sitio web y proceder a la descarga de los circuitos que se quieran analizar:

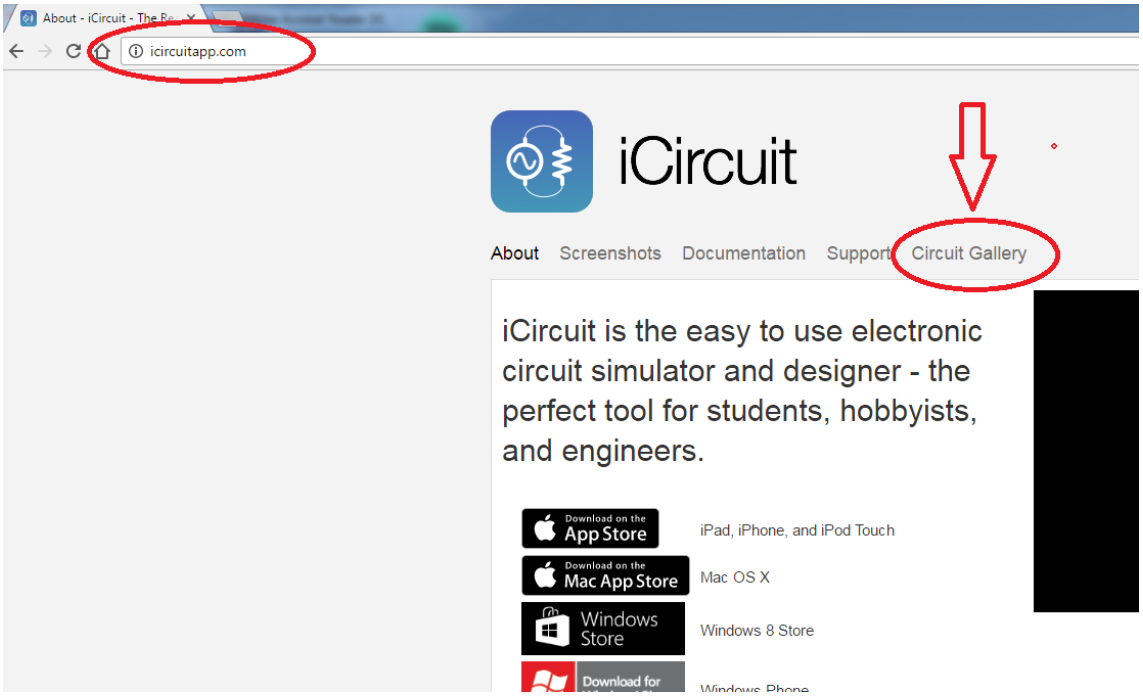


Figura 5.1: *Sitio web oficial de la app iCircuit.*

Una vez que se ha accedido a la galería de circuitos (*Circuit Gallery*), se descargan aquellos sobre los que queramos extraer información. Cuando se selecciona, en la parte derecha aparecerá en verde *Download Circuit*. Este enlace descargará el circuito en formato iCircuit para que sea leído por nuestra aplicación en el caso de tenerla descargada. Por ello, se selecciona el enlace que aparece justo debajo: *Download SVG File*[2].

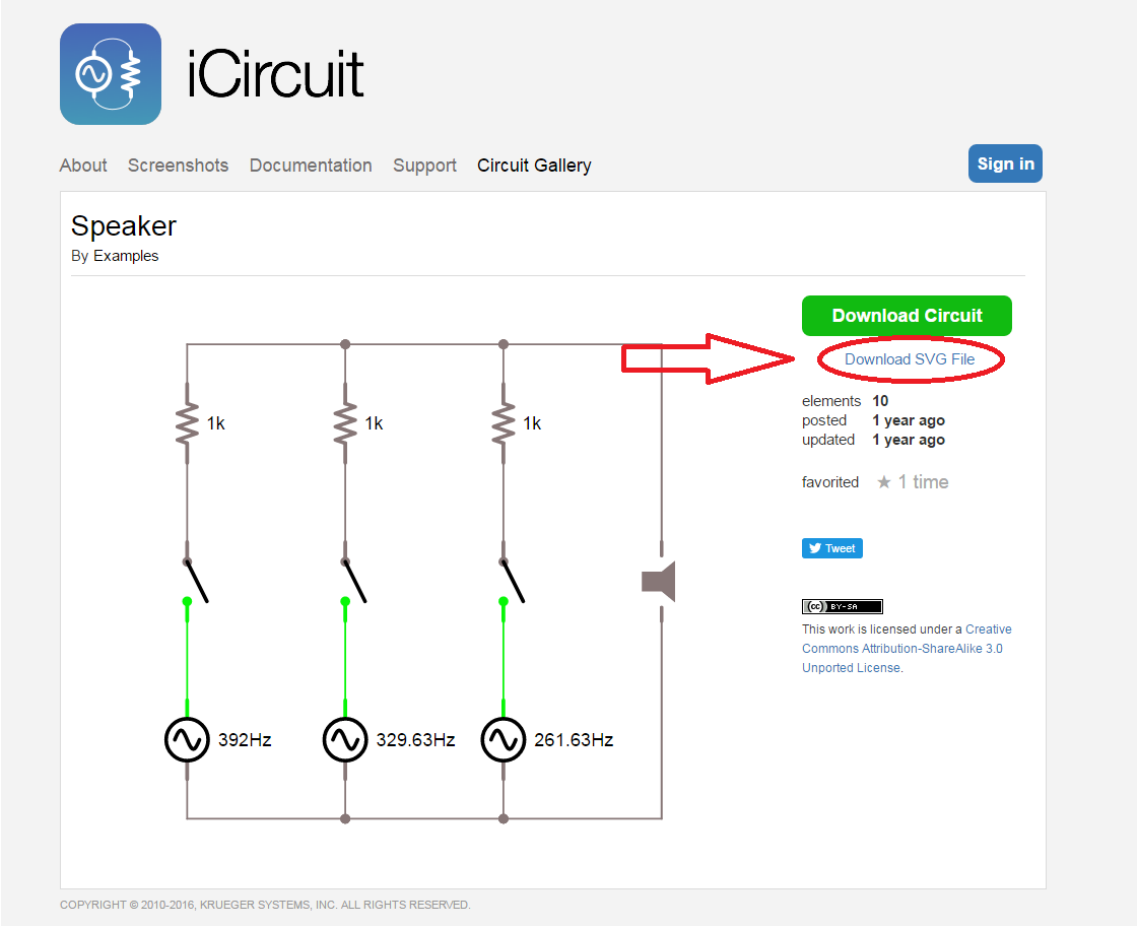


Figura 5.2: *Acceder al circuito.*

Para poder abrir el archivo SVG es necesario algún programa que trabaje con imágenes vectoriales, como puede ser *Adobe Ilustrator*. Sin embargo, hay una alternativa para acceder a dicha imagen sin necesidad de tal programa. Haciendo click al botón derecho, seleccionando Imprimir, y guardando el archivo en versión *pdf*. Una vez hecho esto, se abre en cualquier programa que trabaje con imágenes como Adobe Photoshop y se vuelve a guardar el archivo, esta vez con extensión *png*.

[2] Un archivo SVG es una imagen vectorial.

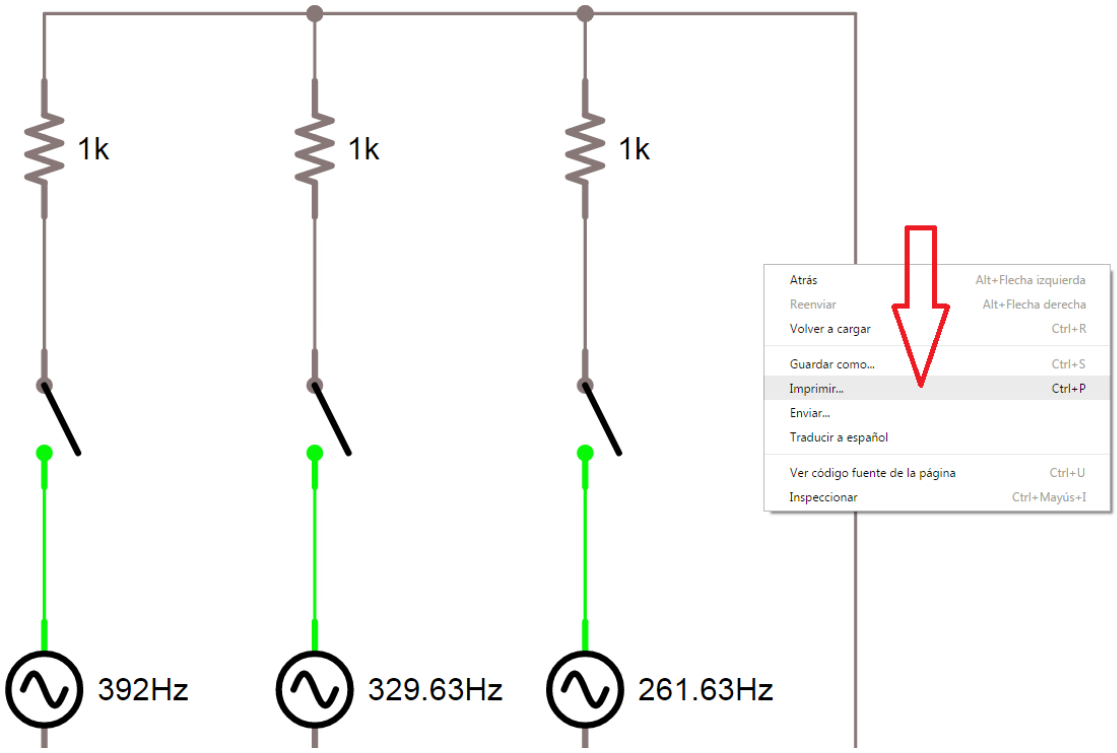


Figura 5.3: *Descargar el circuito.*

5.2 Obtención de resultados

Una vez que se hayan descargado las imágenes, se introducen en la aplicación de extracción de información. Una de las ventanas mostrará el circuito original con todos los elementos que sean detectados introducidos en rectángulos de colores (cada elemento con un color diferente, para que sea más ameno a la vista). Indicar que mostrará todos los elementos, ya sean conocidos o desconocidos, indicándolo en todo momento.

A pesar de que los resultados obtenidos se han conseguido a partir del código implementado en el capítulo anterior, se han introducido algunas modificaciones únicamente de carácter estético, como la forma en que se muestran los nombres de los elementos.

De la *Figura 5.3* se han obtenido los siguientes resultados:

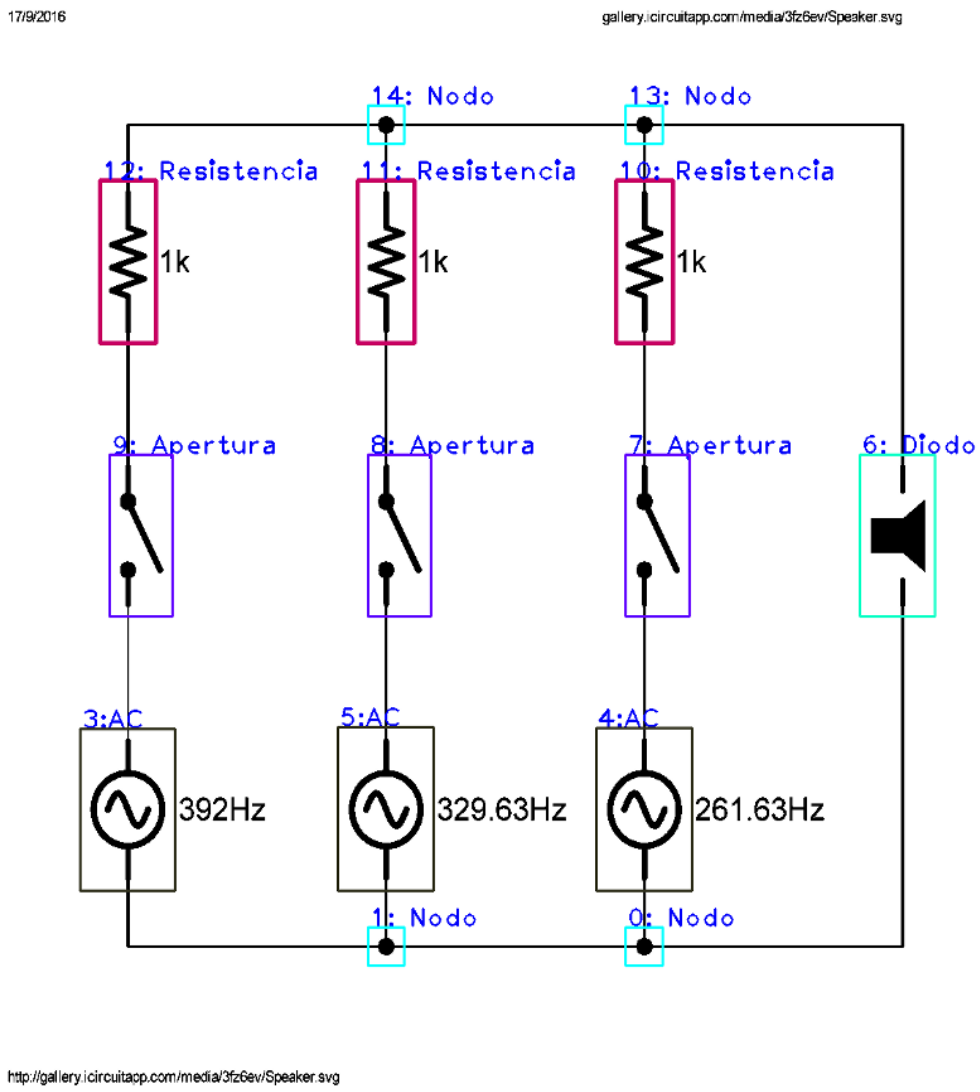


Figura 5.4: *Resultado de la detección de elementos de la Figura 5.3.*

Como se observa, detecta con éxito todos los elementos del circuito, metiéndolos en rectángulos e indicando en la parte superior qué elemento es y qué número representa del total. El único elemento que no detecta correctamente aquí es el **6: Diodo**; este elemento es en realidad un Speaker.

Además, muestra por pantalla la lista de los elementos y el número de ellos que ha reconocido y que le son desconocidos:

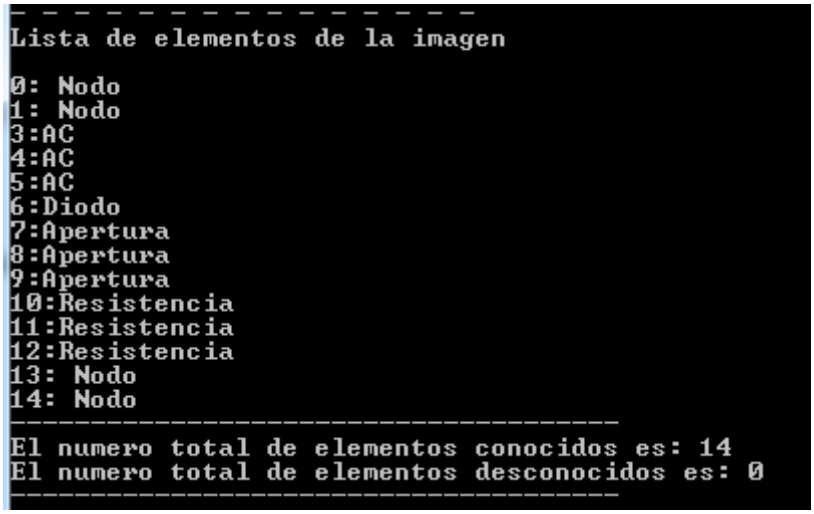


Figura 5.5: *Lista de elementos encontrados en la Figura 5.3.*

Además, el programa mostrará cómo están conectados los elementos (*Figura 5.6*). Indicará qué contornos están unidos por un cable, de tal manera que cruzando esa información con la lista de los elementos se podría dibujar el circuito completo: ése es el objetivo del proyecto, poder dibujar el circuito electrónico completo a partir de la información extraída.

Figura 5.6: *Lista de las conexiones encontradas en la Figura 5.3.*

El programa se ha aprobado en más de cuarenta circuitos descargados directamente de la página oficial de la aplicación de Apple, y en más de ochenta circuitos diseñados en el iPad por mí.

En los circuitos diseñados por mí, el índice de aciertos ha sido del 100%; en los circuitos descargados de internet, aunque el ratio es bueno también, no llega al 100%. Para explicar esta diferencia, se mostrarán a continuación imágenes de resultados de más circuitos donde se ha producido algún error de extracción de información. Después se explicará el porqué de estos errores.

- ***Ejemplo 1***

Se va a extraer la información del circuito de la *Figura 5.7*:

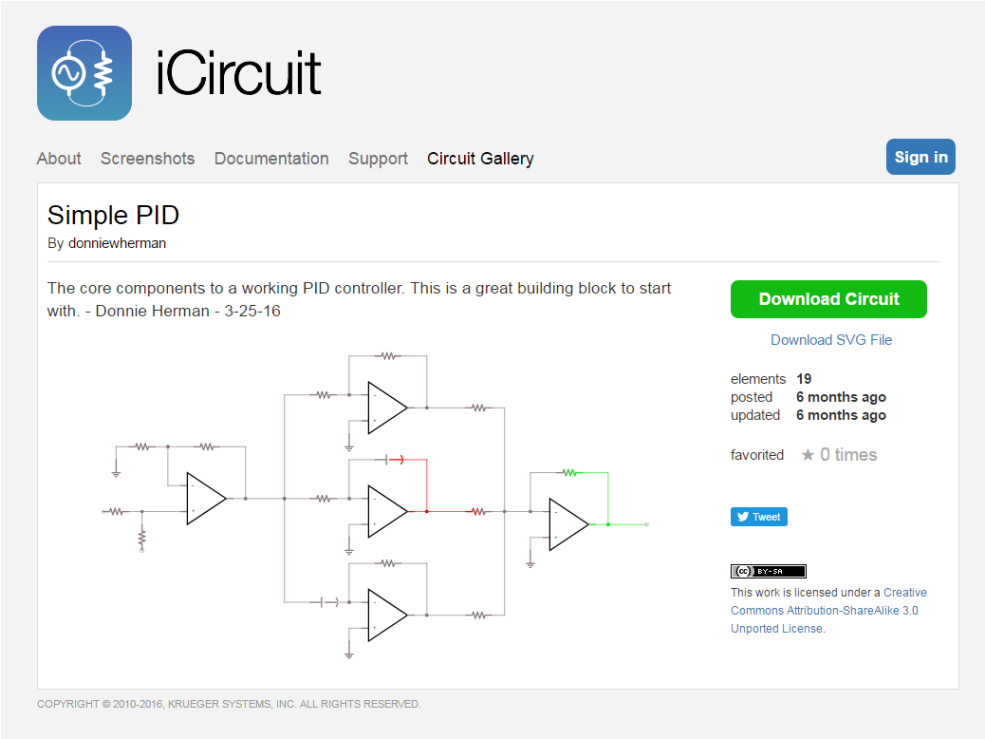


Figura 5.7: *Circuito Simple PID.*

El resultado obtenido en la identificación de los elementos es el mostrado en la *Figura 5.8*:

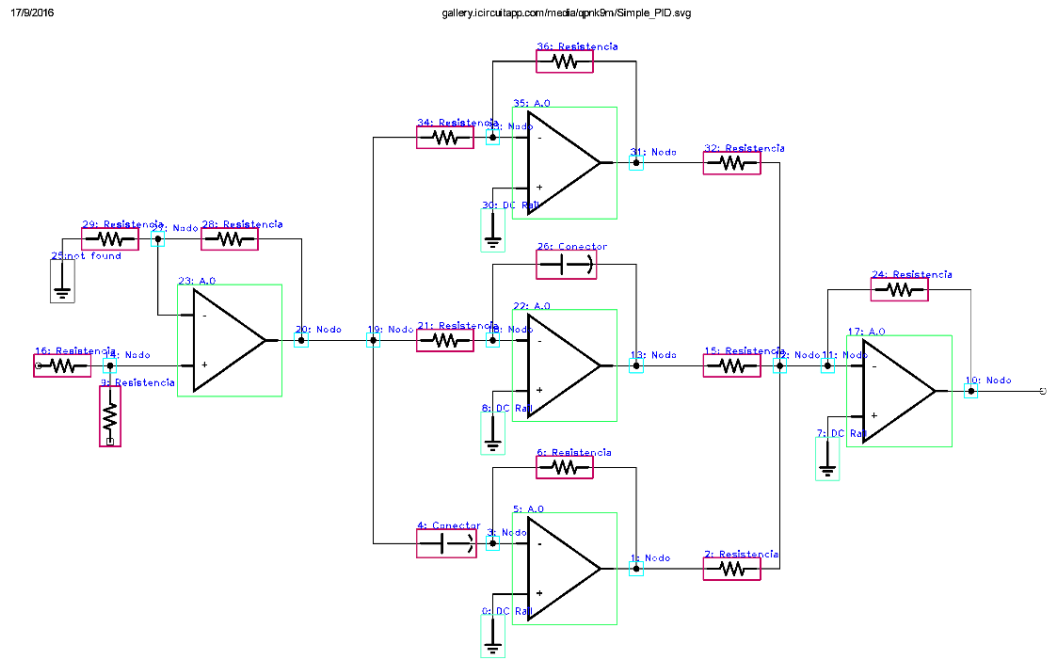


Figura 5.8: *Resultado de la detección de elementos de la Figura 5.7.*

La lista de elementos es la mostrada en la *Figura 5.9*:

```
- - - - - - - - - - - - - - - - - -
Lista de elementos de la imagen

0:DC Rail
1: Nodo
2:Resistencia
3: Nodo
4:Conector
5:Amplificador Operacional
6:Resistencia
7:DC Rail
8:DC Rail
9:Resistencia
10: Nodo
11: Nodo
12: Nodo
13: Nodo
14: Nodo
15:Resistencia
16:Resistencia
17:Amplificador Operacional
18: Nodo
19: Nodo
20: Nodo
21:Resistencia
22:Amplificador Operacional
23:Amplificador Operacional
24:Resistencia
25:not found
26:Conector
27: Nodo
28:Resistencia
29:Resistencia
30:DC Rail
31: Nodo
32:Resistencia
33: Nodo
34:Resistencia
35:Amplificador Operacional
36:Resistencia
---------------------------------------
El numero total de elementos conocidos es: 36
El numero total de elementos desconocidos es: 1
---------------------------------------
```

Figura 5.9: *Lista de elementos encontrados en la Figura 5.7.*

Aquí ha encontrado un total de 36 elementos conocidos y 1 elemento desconocido. Sin embargo, en realidad de esos elementos conocidos, tres de ellos han sido un falso positivo. Como se muestra en la *Figura 5.9*, los tres elementos que ha identificado como *DC Rail*, son en realidad tres *Tierras*; el elemento desconocido también es una *Tierra*. Esto ocurre porque el programa está diseñado para identificar los elementos sin modificar, es decir, usándolos tal y como se crean en la app *iCircuit*.

Cuando se diseña un circuito en dicha aplicación, se seleccionan los elementos que se quieran usar y se arrastran a la interfaz de diseño. Una vez que están los elementos en la interfaz de diseño, se unen mediante cables. Sin embargo, estos elementos pueden alterar su patrón inicial, haciéndolos más pequeños o más grandes (*Figura 5.11*). Al alterar su forma inicial, el programa no es capaz de detectarlos correctamente, por lo que dará error: o lo confunde con otro elemento o indica una mala identificación.

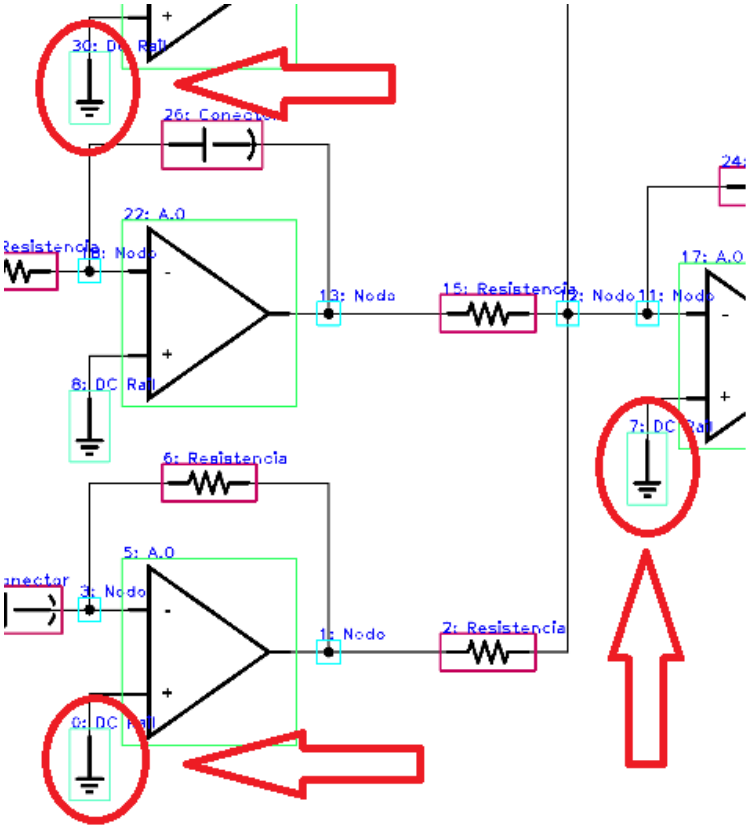


Figura 5.10: *Errores de identificación de la Figura 5.7.*

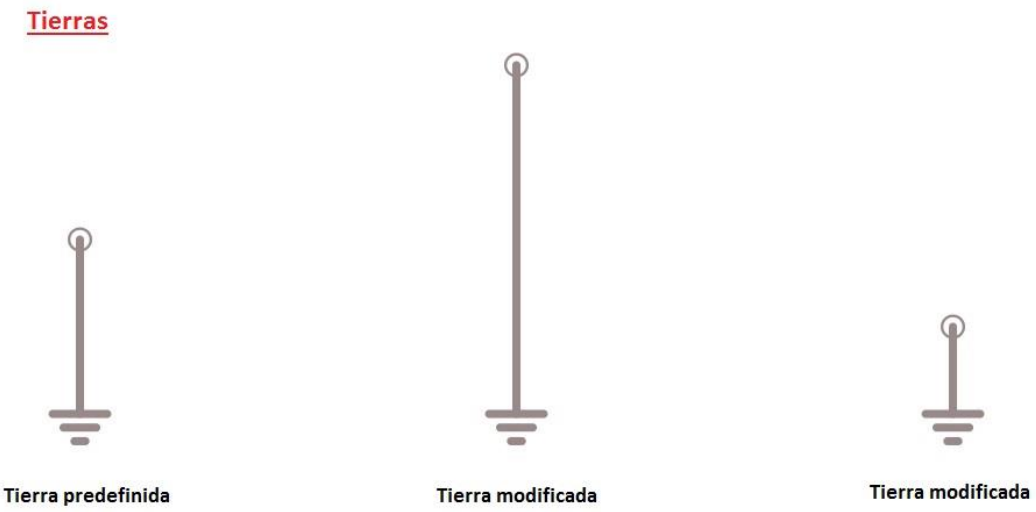


Figura 5.11: *Tierra predefinida y Tierras modificadas.*

Las conexiones del circuito son las de la *Figura 5.12*:

```
 - - CONEXIONES DEL CIRCUITO - -

El contorno 36 y el contorno 33
El contorno 36 y el contorno 31
El contorno 34 y el contorno 19
El contorno 34 y el contorno 33
El contorno 33 y el contorno 35
El contorno 35 y el contorno 31
El contorno 31 y el contorno 32
El contorno 32 y el contorno 12
El contorno 29 y el contorno 25
El contorno 29 y el contorno 27
El contorno 27 y el contorno 28
El contorno 28 y el contorno 20
El contorno 27 y el contorno 23
El contorno 26 y el contorno 18
El contorno 26 y el contorno 13
El contorno 24 y el contorno 11
El contorno 24 y el contorno 10
El contorno 23 y el contorno 20
El contorno 20 y el contorno 19
El contorno 19 y el contorno 21
El contorno 21 y el contorno 18
El contorno 18 y el contorno 22
El contorno 19 y el contorno 4
El contorno 16 y el contorno 14
El contorno 14 y el contorno 23
El contorno 22 y el contorno 13
El contorno 13 y el contorno 15
El contorno 15 y el contorno 12
El contorno 12 y el contorno 11
El contorno 11 y el contorno 17
El contorno 14 y el contorno 9
El contorno 17 y el contorno 10
El contorno 17 y el contorno 7
El contorno 6 y el contorno 3
El contorno 6 y el contorno 1
El contorno 4 y el contorno 3
El contorno 3 y el contorno 5
El contorno 5 y el contorno 1
El contorno 1 y el contorno 2
- - - - - - - - - - - - - - - -
```

Figura 5.12: *Lista de las conexiones encontradas en la Figura 5.7.*

- ***Ejemplo 2***

Se va a extraer la información del circuito de la *Figura 5.13*:

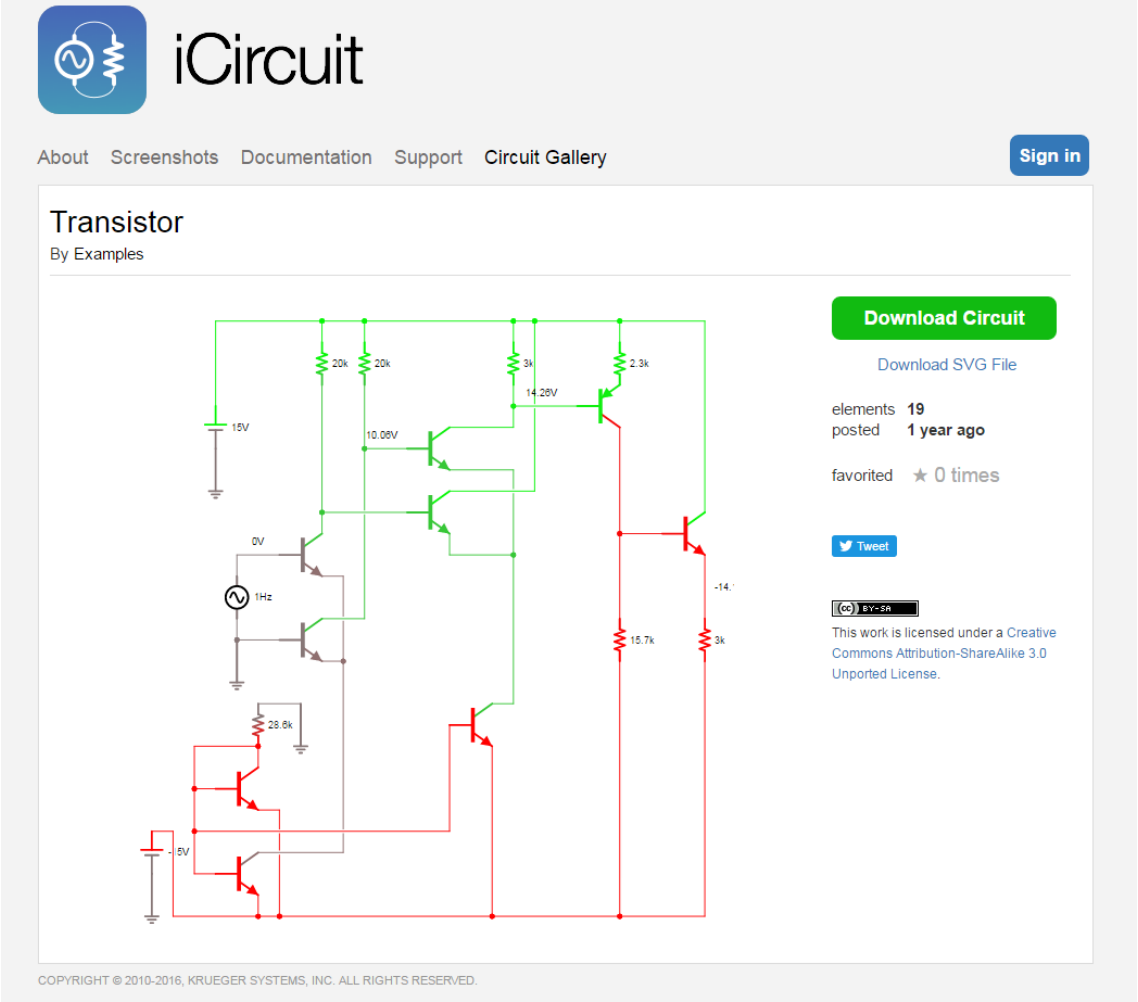


Figura 5.13: *Circuito Transistor.*

El resultado obtenido en la identificación de los elementos es el mostrado en la *Figura 5.14*:

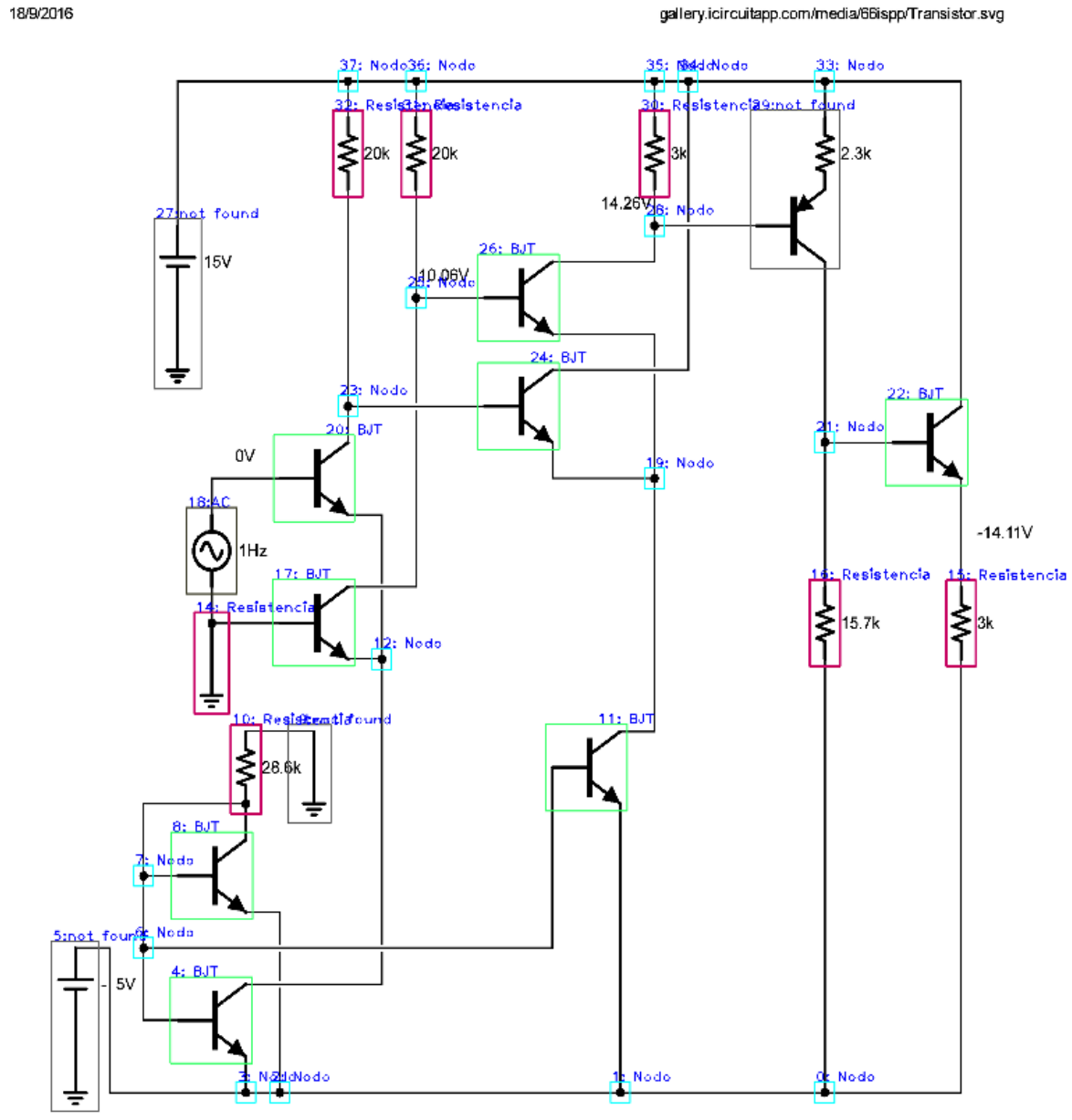


Figura 5.14: *Resultado de la detección de elementos de la Figura 5.7.*

La lista de elementos es la mostrada en la *Figura 5.15*:

Figura 5.15: *Lista de elementos encontrados en la Figura 5.14.*

Aquí aparece un nuevo fallo de identificación de elementos. El programa es capaz de identificar correctamente todos los elementos siempre y cuando se hallen aislados, es decir, unidos mediante cables, y no uniendo directamente un elemento a otro. Cuando aparecen dos elementos unidos, el programa entiende que es un único elemento, por lo que fallará en la identificación. Esto se muestra en la *Figura 5.16, Figura 5.17 y Figura 5.18*:

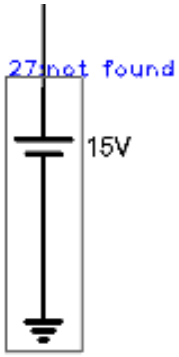


Figura 5.16: *DC + Tierra.*

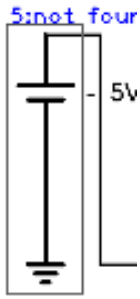


Figura 5.17: *DC + Tierra.*

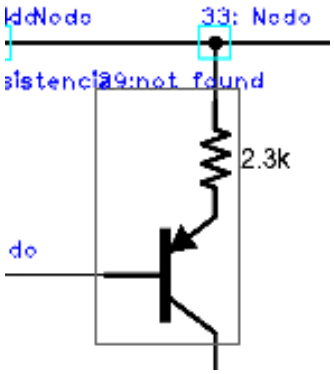


Figura 5.18: *BJT + Resistencia.*

Las conexiones del circuito son las de la *Figura 5.19*:

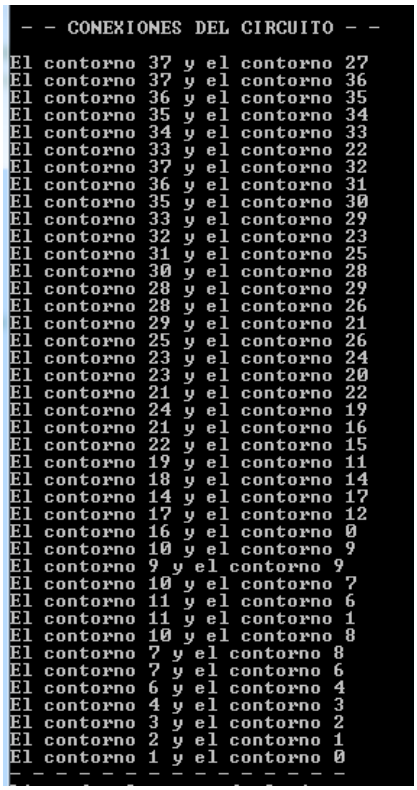


Figura 5.19: *Lista de las conexiones encontradas en la Figura 5.13.*

En la parte final de la memoria se incluirán varios ejemplos más de circuitos (*anexo ejemplos circuitos*). Lo que se mostrará a continuación es una tabla donde se recogerán todos los aciertos y fallos del programa a la hora de extraer información de los diagramas de circuitos electrónicos. Con ello se pretende reflejar el ratio de aciertos – fallos del proyecto.

La *Tabla 5.1* muestra los porcentajes de elementos conocidos y desconocidos totales de los circuitos más representativos, y cuáles han sido falsos positivos, para comprobar la exactitud del programa.

Todos los falsos positivos se deben a lo mencionado con anterioridad: elementos modificados o elementos unidos.

Circuitos	Elementos conocidos	Elementos desconocidos	Falsos positivos	Ratio de aciertos
Amplificador0	36	1	4	88,90%
Amplificador1	23	8	3	90,32%
basico0	4	2	2	66,67%
bjt0	8	1	1	88,90%
bjt1	4	3	4	42,86%
bjt2	10	4	5	64,28%
bjt3	14	0	1	92,86%
bjt4	7	0	1	85,72%
diodo0	16	1	3	82,35%
speaker	14	0	1	92,86%
src	3	4	3	57,14%
transformador0	3	1	1	75%
transformador1	28	3	4	87,10%
transformador3	23	4	2	92,60%
transistor	33	4	1	97,30%
	Elementos conocidos totales	Elementos desconocidos totales	Falsos positivos totales	Ratio de aciertos total
	226	36	36	80,35%

Tabla 5.1: *Ratios de aciertos identificación de los elementos.*

Capítulo 6. CONCLUSIONES Y TRABAJOS FUTUROS

La aplicación de extracción de la información presente en diagramas en formato imagen –de circuitos electrónicos analógicos- ha obtenido un resultado óptimo en cuanto a los objetivos establecidos.

El programa es capaz de analizar cualquier esquema de circuito electrónico siempre y cuando use los elementos básicos citados. Los datos de este análisis se muestran en dos partes: una imagen del circuito original –el que ha sido introducido en la aplicación- con los nombres de todos los elementos escritos, y en formato texto cómo se conectan los elementos encontrados en el circuito. Además muestra cuántos han sido los elementos de cada tipo, así como el número total de ellos.

Al ser este proyecto un prototipo de OCR para diagramas, son muchas las futuras aplicaciones que pueden surgir a partir de aquí.

Algunas de ellas podrían ser las siguientes:

- **Comunicación con programas de electrónica:** una vez que esta aplicación ha realizado el análisis de un circuito, podría enviar la información a un programa de diseño de circuitos electrónicos. El programa de diseño, a partir de esta información, sería capaz de construir el circuito. Posteriormente, sería capaz de editar, cambiar valores, etc.

- **Aplicación a diversos tipos de diagramas:** al igual que se ha hecho en este proyecto, al usar una plantilla maestra, se podría realizar la misma acción para cualquier tipo de diagramas. En dicha plantilla, se introducirían los elementos que compongan el diagrama alternativo, y después la aplicación debería ser capaz de analizar las imágenes a partir de él.

- **Creación de una base de datos de circuitos:** se podría crear una base de datos con una cantidad enorme de diagramas de circuitos. El objetivo sería que sirviese de guía para cualquiera que necesitara trabajar con diagramas de circuitos; accedería a dicha base de datos y, filtrando la búsqueda a sus intereses, descargaría tantos ejemplos de circuitos como quisiera. En este caso se podría filtrar la búsqueda por palabras y números clave, por ejemplo: *2 Tierra – 1 Voltaje – 5 Resistencia – 10 Circuito*, y el programa devolvería los diez primeros circuitos que encuentre que tengan dos tierras, un voltaje y cinco resistencias. Lo ideal sería que se exportasen de la base de datos directamente a un programa de electrónica (dicho programa podría llegar a un acuerdo con la base de datos e incluir esta opción en su propia interfaz, de manera que estén conectados programa y base de datos), para poder trabajar con ellos.

Finalmente, un área de mejora de este trabajo sería crear variaciones para otros programas diferentes del aquí utilizado, iCircuit, explorando nuevas posibilidades y líneas de investigación en el desarrollo de la extracción de la información de circuitos electrónicos.

CAPITULO 7. PRESUPUESTO Y MARCO LEGAL

7.1. Presupuesto

En el **presupuesto** se van a incluir todos los elementos que han sido necesarios para la realización de este proyecto, así como una estimación de los costes. Se van a dividir los costes en dos grupos:

- Costes de personal

- Costes de material

Para el cálculo de dichos costes se tendrán en cuenta que: todos los costes estarán reflejados en Euros, y se usarán dos decimales para el redondeo de los valores.

7.1.1. Costes de personal:

Se reflejará el coste unitario por hora de desarrollo en función del sueldo de un becario. Se estimará que el tiempo invertido para desarrollar este proyecto ha sido de 23 días, a tiempo completo; o un mes laboral de 31 días.

Ítem	Descripción técnica de equipos y materiales requeridos	Q	Precio (material + instalación)	Precio Total
1	Horas de desarrollo	184	6.25 €	1150 €

Tabla 6.1: *Costes de personal.*

7.1.2. Costes de material:

Ítem	Descripción técnica de equipos y materiales requeridos	Q	Precio (material + instalación)	Precio Total
1	Hp Z200 Workstation. 4 GB de memoria RAM. Procesador Intel Core i3 a 3.20 GHz.	1	500 €	500 €
2	Sistema Operativo Windows 7	1	140 €	140 €
3	Microsoft Visual Studio 2015	1	400 €	400 €
4	Librería OpenCV	1	0 €	0 €
5	Internet	1	100 €	100 €
6	Pantalla Hp ZR22w	2	150 €	300 €
7	Teclado Hp	1	15 €	15 €
8	Ratón Hp	1	10 €	10 €
9	ICircuit	1	10 €	10 €
	TOTAL			**1465 €**

Tabla 6.2: *Costes de material.*

El coste podría ser menor si se utilizara un ordenador de menor calidad, siempre y cuando se comprueben antes los requerimientos específicos del Sistema Operativo Windows 7 y del software Microsoft Visual Studio 2015 (15). El coste de las pantallas se podría reducir a sólo una, pero es recomendable el uso de dos ya que aumenta la eficiencia del desarrollo.

El coste de internet se ha calculado en base al contrato de una línea básica, ya que su uso es sólo necesario como fuente de información.

También es necesario para la descarga del Sistema Operativo Windows 7, del software Microsoft Visual Studio 2015 y de la librería OpenCV (16); si se redujese la velocidad de descarga debido a la contratación de otra línea, el coste en horas de desarrollo aumentaría por el tiempo extra necesario para ello.

El Sistema Operativo Windows 7 y el software Microsoft Visual Studio 2015 son de uso gratuito sólo con fines académicos o de investigación. Por lo que si se quisiera realizar algún otro uso de éstos, su coste ya no sería de cero euros.

7.1.3. Coste Total

Costes de personal	**1150€**
Costes de material	**1465€**
COSTE TOTAL	**2615€**

Tabla 6.3: *Coste Total.*

7.2. Marco Legal

En cuanto al **marco legal**, se debe tener en cuenta los siguientes aspectos.

Si se quisiera hacer un uso no académico del proyecto, como comercializarlo, se debe contratar Microsoft Visual Studio Professional 2015 (coste: 646.00 € incluido I.V.A), Microsoft Visual Studio Professional con MSDN (coste: 1035.00 € incluido I.V.A), Microsoft Visual Studio Test Professional con MSDN (coste: 1165.00 € incluido I.V.A), o Microsoft Visual Studio Enterprise con MSDN (coste: a partir de 3328.00 € incluido I.V.A).

La librería OpenCV es totalmente gratuita y permite un uso tanto académico como comercial, siempre y cuando se especifique en el producto.

Aun así, antes de hacer un uso no académico de este proyecto, es conveniente consultar a una persona profesional en el ámbito legal.

CAPITULO 8. BIBLIOGRAFIA

1. Gardey, Julián Pérez Porto y Ana. http://definicion.de/informacion/. 2012.

2. Medina, Lic. Leslie Slazar. http://www.monografias.com/trabajos82/historia-escritura/historia-escritura.shtml.

3. https://es.wikipedia.org/wiki/Informaci%C3%B3n#Bibliograf.C3.ADa. 4. Medino, Julián Pérez Porto y María. http://definicion.de/imagen-digital/. 2014.

4. Ministerio de Educación, Gobierno de España. http://www.ite.educacion.es/formacion/materiales/86/cd/pdf/m2_caracteristicas_de_la_imagen_digital.pdf.

5. Escalera, Arturo de la y Armingol, José María. http://ocw.uc3m.es/ingenieria-de-sistemas-y-automatica/sistemas-de-percepcion/transparencias-clase/5-preprocesamiento.pdf. 16 de 07 de 2010.

6. García, Dr. Nicolás Luis Fernández. http://www.uco.es/users/ma1fegan/2012-2013/vision/Temas/segmentacion.pdf.

7. Provencio, Francisco Recio y David. http://www.desarrolloweb.com/articulos/1328.php. 10 de 12 de 2003.

8. Caminos, Gonzalo. https://hipertextual.com/analisis/visual-studio-2015. 20 de 9 de 2015.

9. https://msdn.microsoft.com/es-es/library/ee822860(v=vs.100).aspx.

10.http://www.cuatrorios.org/index.php?option=com_content&view=article&id=169:opencv-librer%C3%ADa-de-visi%C3%B3n-por-computador&catid=39:blogsfeeds. 24 de 11 de 2016.

11. V. M. Arévalo, J. González, G. Ambrosio. http://mapir.isa.uma.es/varevalo/drafts/arevalo2004lva1.pdf.

12. http://icircuitapp.com/.

13. https://www.visualstudio.com/products/.

14. http://opencv.org/documentation.html.

15. https://hipertextual.com/analisis/visual-studio-2015.

16. http://www.desarrolloweb.com/articulos/1328.php.

17.http://www.cuatrorios.org/index.php?option=com_content&view=article&id=169:opencv-librer%C3%ADa-de-visi%C3%B3n-por-computador&catid=39:blogsfeeds.

18. https://msdn.microsoft.com/es-es/library/hh875011.aspx.

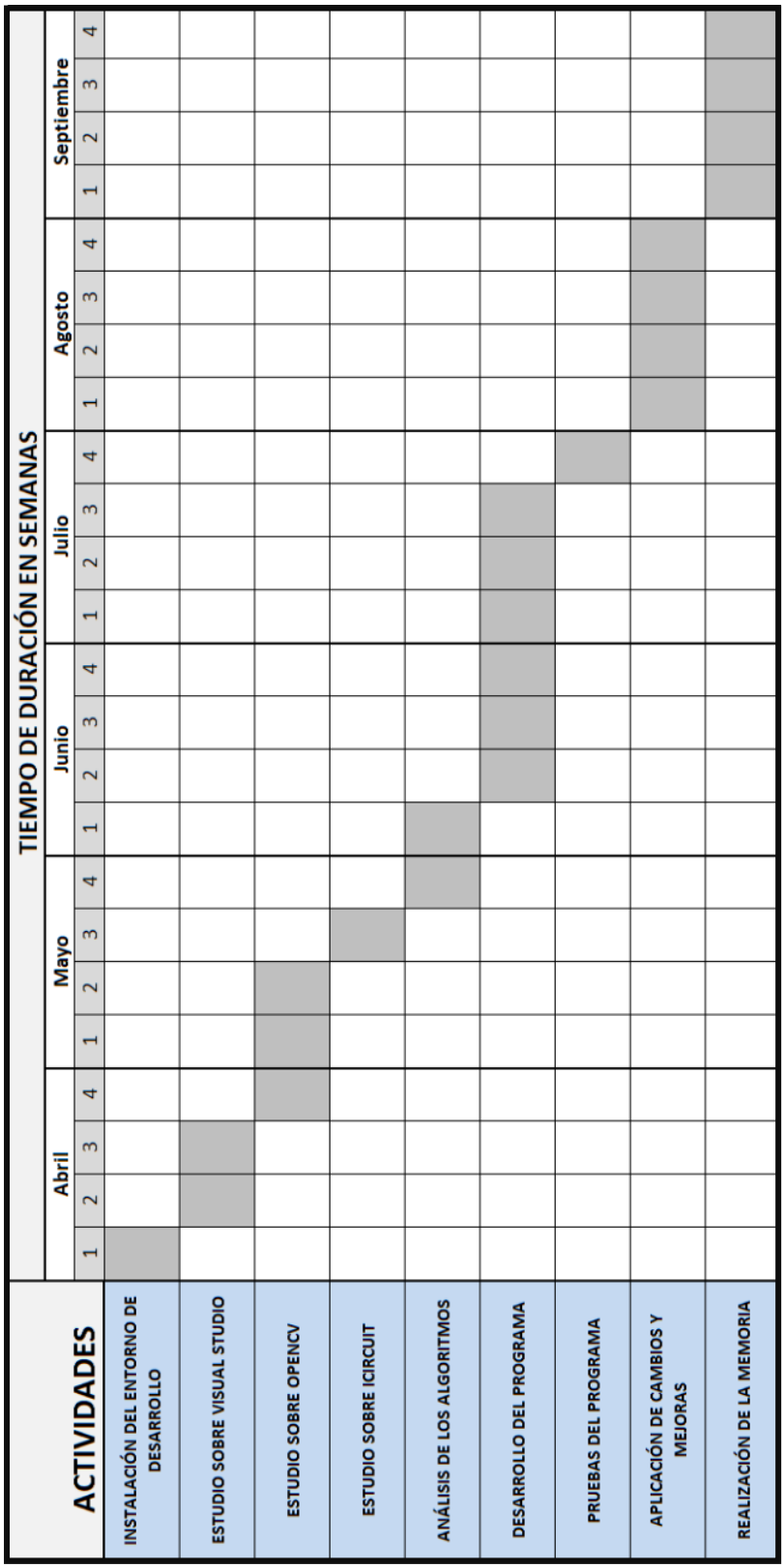
INDICE DE FIGURAS Y TABLAS

FIGURAS

TABLAS

ANEXO I

Diagrama de Gantt del proyecto



ANEXO II

Instalación Visual Studio 2015

En primer lugar, se deberá acceder a la web oficial de Microsoft Visual Studio (https://www.visualstudio.com/es-es/downloads/download-visual-studio-vs.aspx); se podrá descargar de manera gratuita Visual Studio.

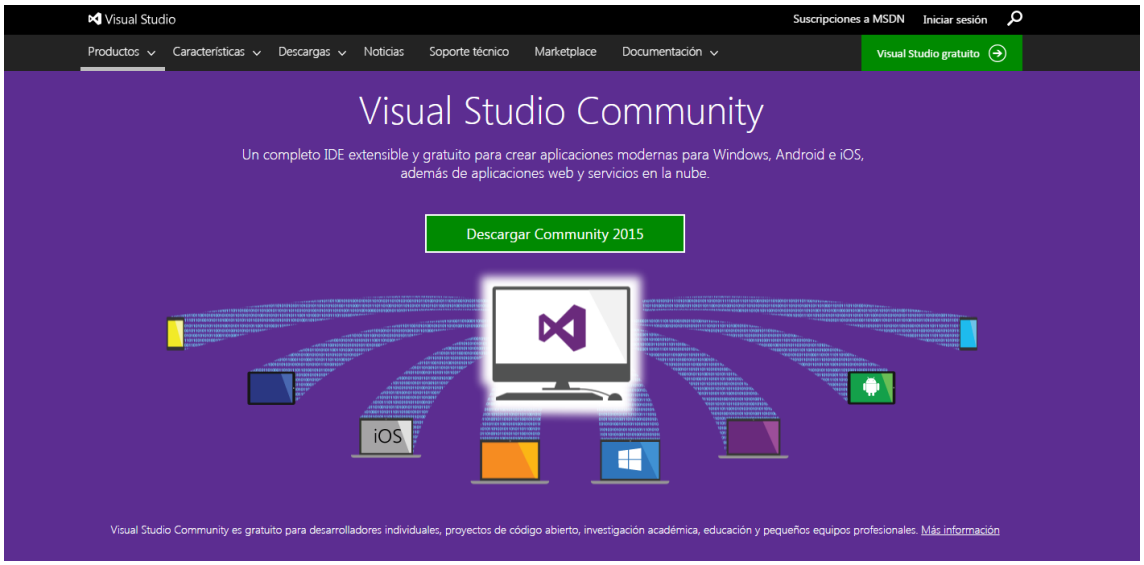


Figura 1: *Web oficial de Microsoft Visual Studio.*

Una vez descargado, abrimos el fichero y pulsaremos "Ejecutar", comenzando en ese momento la instalación. Se abrirá la ventana de instalación de Visual Studio Community 2015, en la que se analizaran los requerimientos del sistema, indicándonos si faltara algo (se pueden ver aquí los requisitos mínimos de Visual Studio: https://www.visualstudio.com/es-es/downloads/visual-studio-2015-system-requirements-vs.aspx).

A continuación nos dará la opción de elegir la ubicación de instalación, así como el tipo de instalación: típica o por defecto, y personalizada. Para este proyecto la configuración por defecto es perfectamente viable, sin embargo se procederá a realizar la instalación personalizada, ya que permite instalar muchas más características que podrían ser útiles para una ampliación del proyecto más adelante.

Figura 2: *Instalación 1.*

Se marcarán ahora las peculiaridades que se quieran instalar (Web Developer Tolls, PowerShell para Visual Studio, Silberlight, desarrollo de aplicaciones universales, desarrollo para móviles multiplataforma, etc.) y se pulsará “Siguiente”.

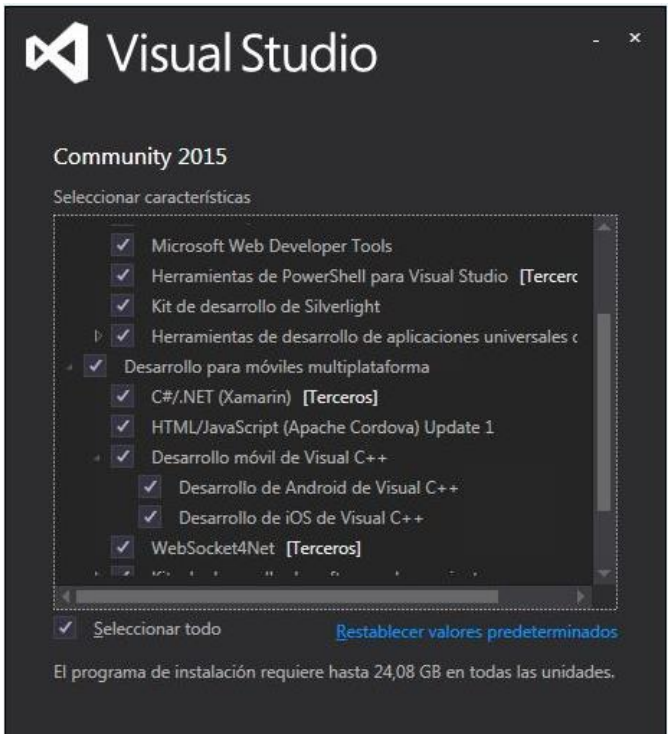


Figura 3: *Instalación 2.*

Leídos los requerimientos de la licencia de las distintas peculiaridades marcadas, y estando de acuerdo se continuara con la instalación de Visual Studio .NET Community 2015 pulsando el botón “Instalar”.

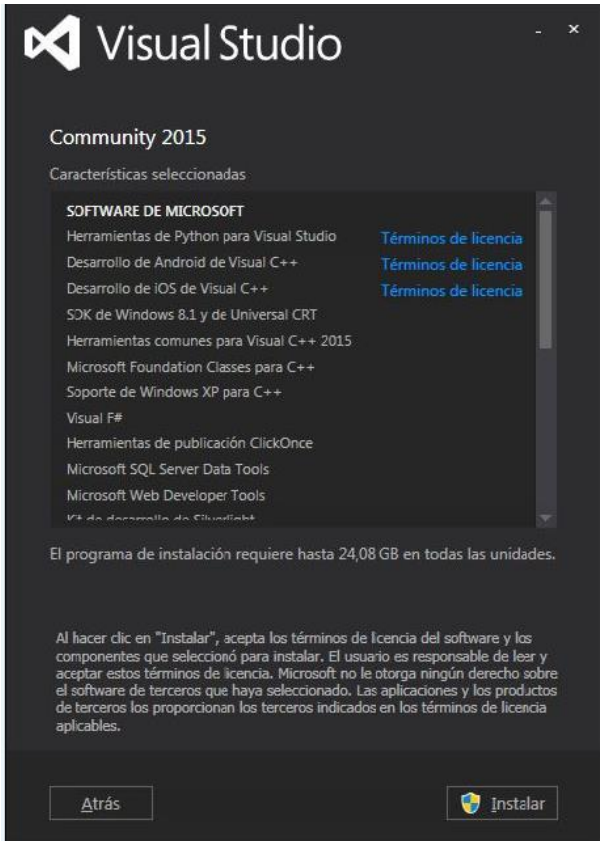


Figura 4: *Instalación 3.*

En este momento empezará la descarga e instalación definitiva de Visual Studio .NET Community 2015:

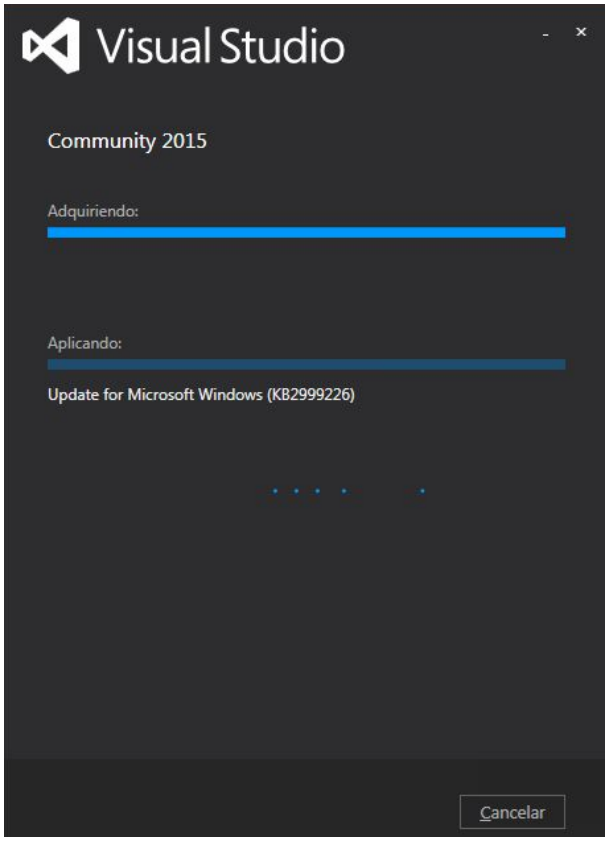


Figura 5: *Instalación 4.*

Transcurrido un periodo de tiempo (depende de las peculiaridades escogidas), el asistente señalará que la instalación ha finalizado. Se deberá reiniciar el equipo pulsando "Reiniciar ahora":

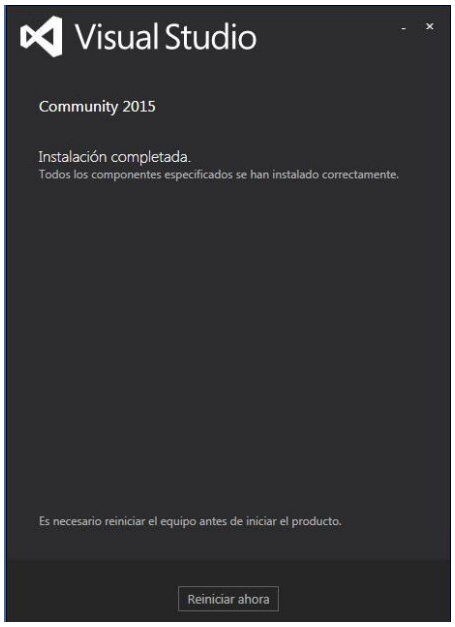


Figura 6: *Instalación 5.*

Una vez instalado, cuando se inicie el programa por primera vez, dará la opción de personalizar la interfaz. Es una opción tan sólo de comodidad, puesto que no varía en nada las opciones del programa; simplemente es para tener personalizado tanto los colores como los lugares de las herramientas. Aun así, una vez iniciado el programa, se podrán volver a cambiar.

Al abrir el programa, se seleccionará "Nuevo proyecto", a la izquierda de la interfaz:

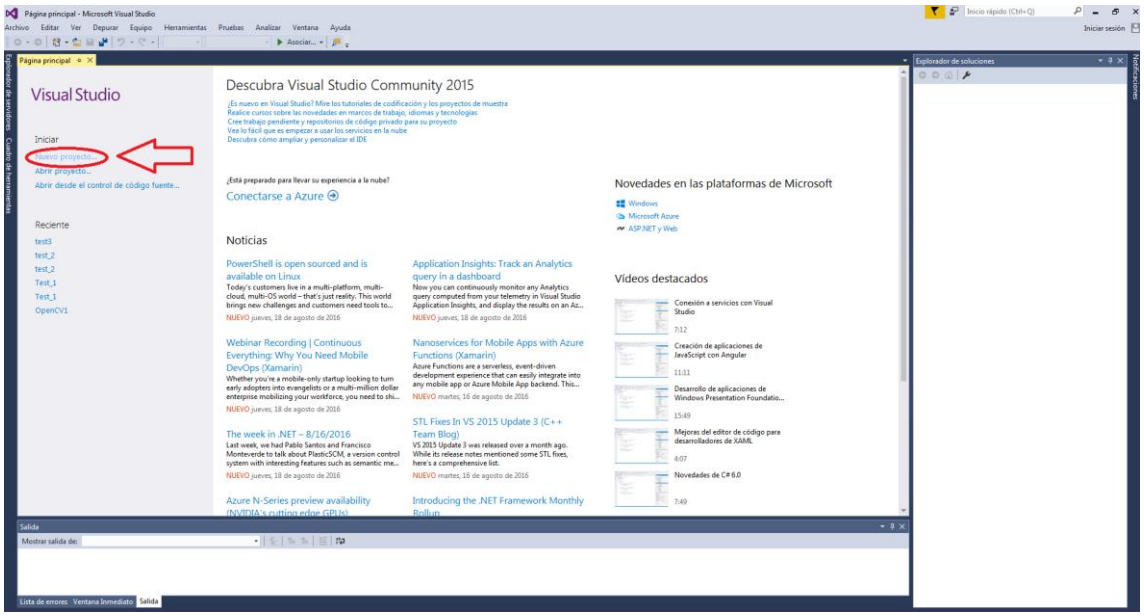


Figura 7: *Instalación 6.*

Se desplegará otro menú en el que se deberá elegir sobre que lenguaje de programación queremos realizar nuestro programa. Dentro de cada tipo de lenguaje se podrá escoger entre varios tipos de aplicaciones. En este caso, se seleccionará el *Visual C++* (es el lenguaje de programación que más controlo y es, además, uno de los más usados); y, dentro de los tipos de aplicaciones que deja elegir, se optará por *Aplicación de consola Win32*.

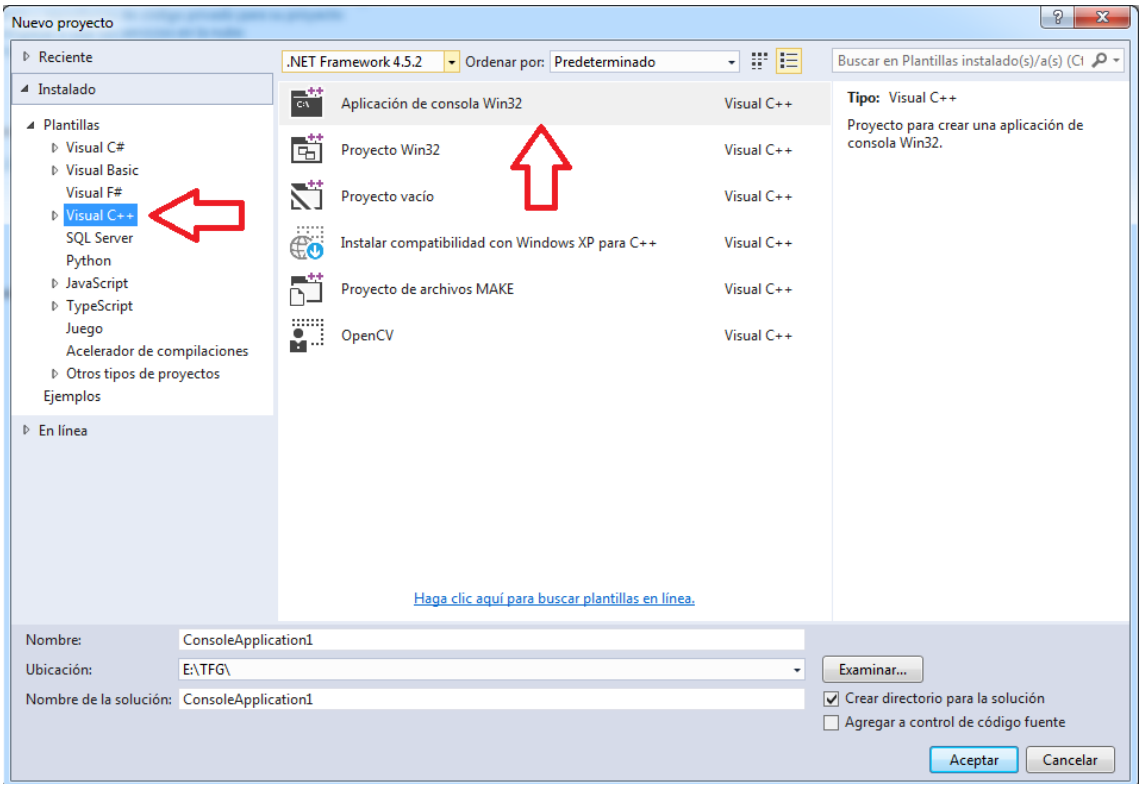


Figura 8: *Instalación 7.*

En la imagen, además de verse los tipos de aplicación por defecto, se ve, como última opción, *OpenCV*. Es la aplicación que realmente se usará, pero para poder hacerlo se debe crear como una Plantilla personalizada que más adelante se explicará. Por ahora, se selecciona *Aplicación de consola Win32* y se da a "Aceptar".

Ahora se deberán configurar las características de la aplicación; se seleccionará *Proyecto vacío* y se pulsa "Finalizar":

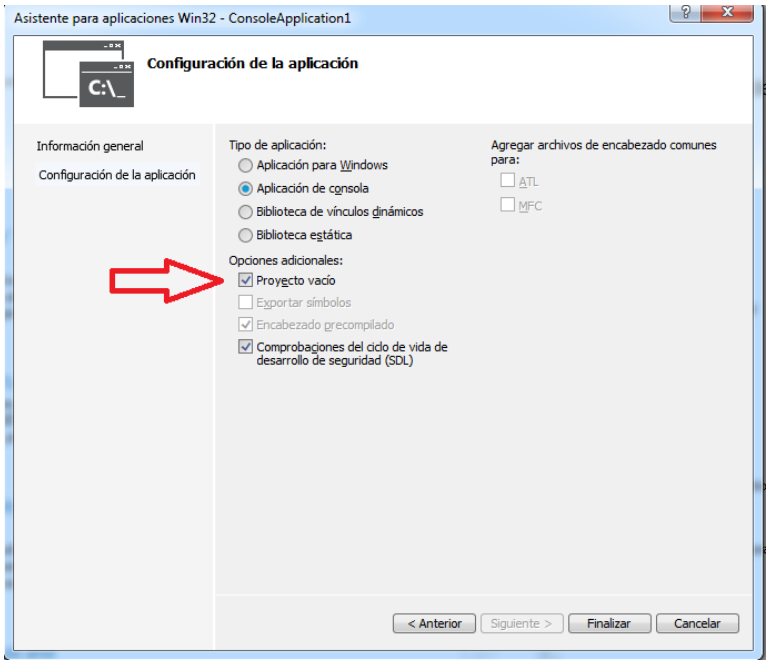


Figura 9: *Instalación 8.*

Una vez hecho todo esto, el proyecto ya estará creado, apareciendo en la pantalla lo siguiente:

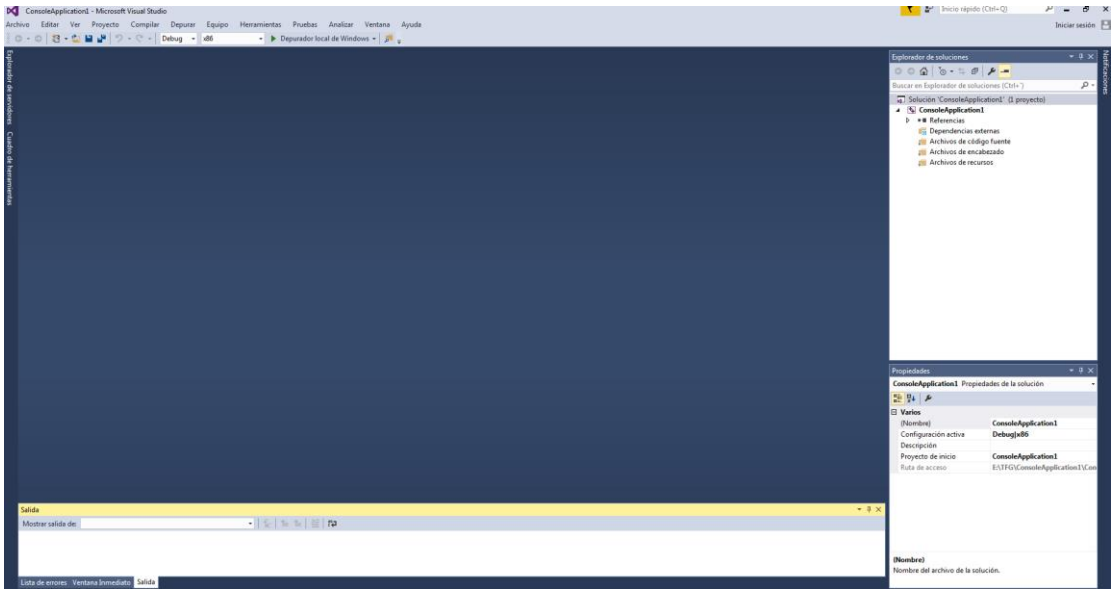


Figura 10: *Instalación 9.*

Aun no se puede comenzar a escribir el código, faltan aún algunos pasos, siendo el más importante añadir la librería OpenCV, que se explicará a continuación.

ANEXO III

OpenCV

Una vez instalado Visual Studio y creado el proyecto nuevo, se deberá incluir la librería OpenCV.

El primer paso es descargarse la librería OpenCV de la página oficial (http://opencv.org/downloads.html); a ser posible la última actualización, ya que siempre incorpora mejoras o nuevas funciones, seleccionando también el sistema operativo acorde. En este caso, se descargará OpenCV para Windows. Una vez descargado, se descomprimirá el archivo.

Se debe prestar especial atención a las direcciones que hay que usar a la hora de configurar (hay que asegurarse de que las rutas hasta los archivos están bien, o de lo contrario el programa no funcionará). La dirección que se ha seleccionado para guardar los archivos de la librería es *C:*\.

Para la configuración se deberá tener en cuenta la siguiente nomenclatura de los archivos descargados: vc9 = Visual Studio 2008, vc10 = Visual Studio 2009, vc12 = Visual Studio 2013 y vc14 = Visual Studio 2015. En este caso, como se tiene instalado Visual Studio 2015, se necesitarán los archivos de la carpeta vc14 que se verá más adelante.

Hay que configurar el Path de Windows siguiendo estas indicaciones: *Panel de control / Sistema / Configuración avanzada del sistema / Variables de entorno;* se pulsa en “Path” y en “Editar”.

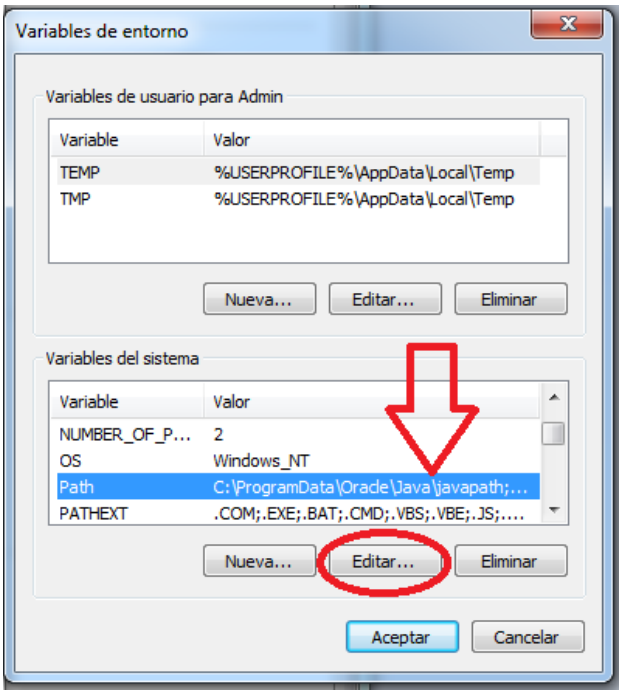


Figura 1: *OpenCV 1.*

Se pulsa "Nuevo" y se añade: *C:\opencv\build\x64\vc14\bin*. Se pulsa "Aceptar":

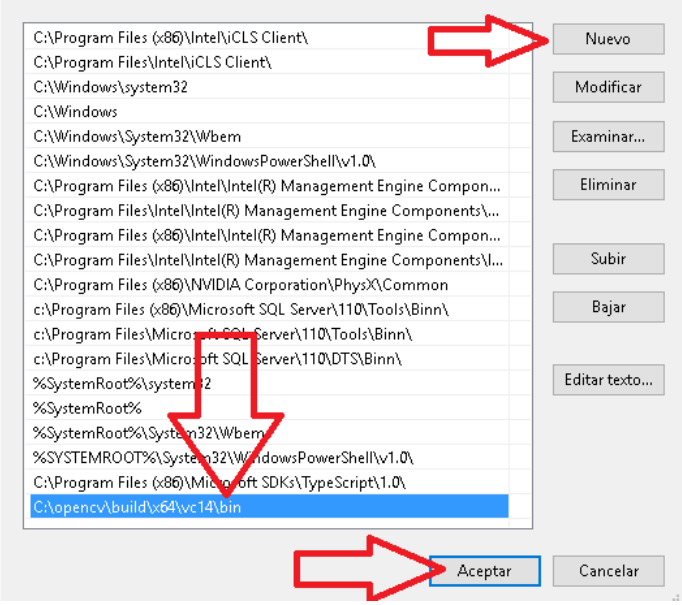


Figura 2: *OpenCV 2.*

Ahora de vuelta otra vez a Visual Studio, al proyecto nuevo que se había creado, se realiza lo siguiente: Encima del nombre del proyecto se selecciona “Propiedades” (con el botón derecho).

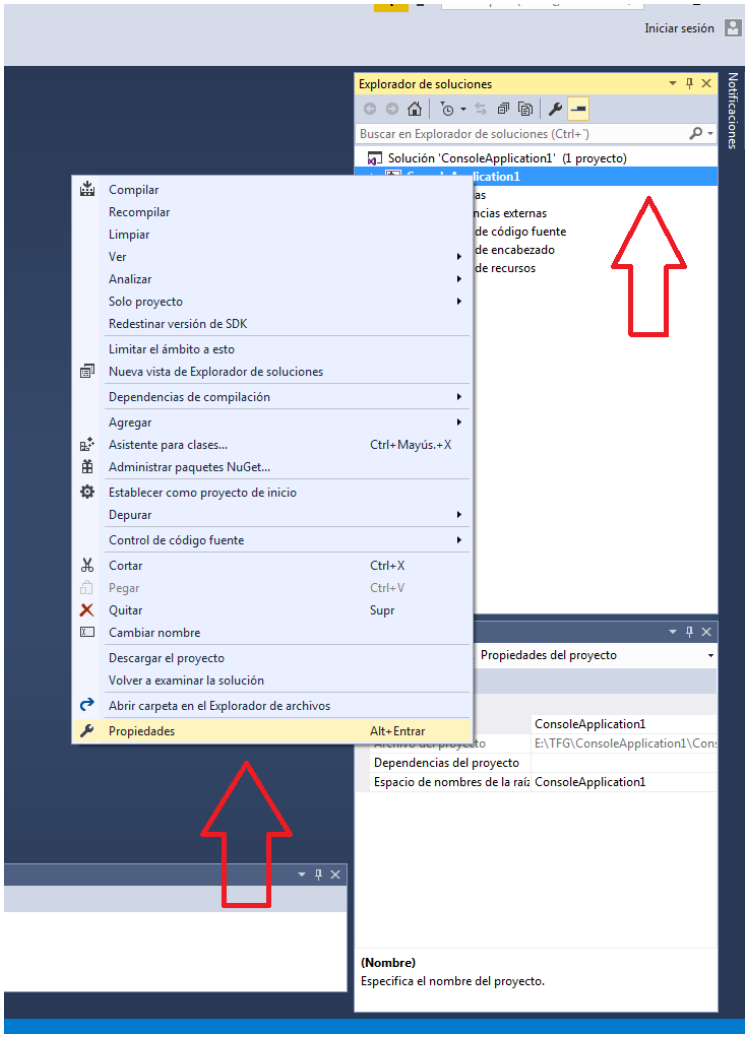


Figura 3: *OpenCV 3.*

Sobre la ventana que se abre (menú de la izquierda), se selecciona *Directorios de VC++*, y en *Directorios de archivos ejecutables* pulsamos en “editar”.

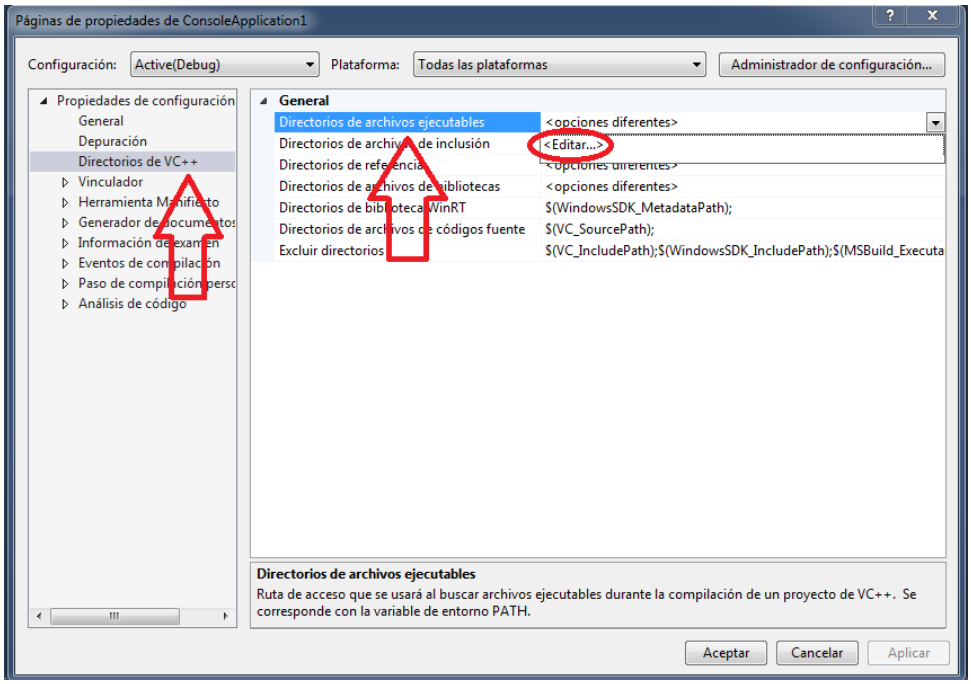


Figura 4: *OpenCV 4.*

Se abre una nueva ventana y se pulsa en el icono de la carpeta para agregar un nuevo directorio. Después se busca el directorio en el icono de puntos suspensivos; se debe agregar el siguiente directorio: *C:\opencv\build\x64\vc14.*

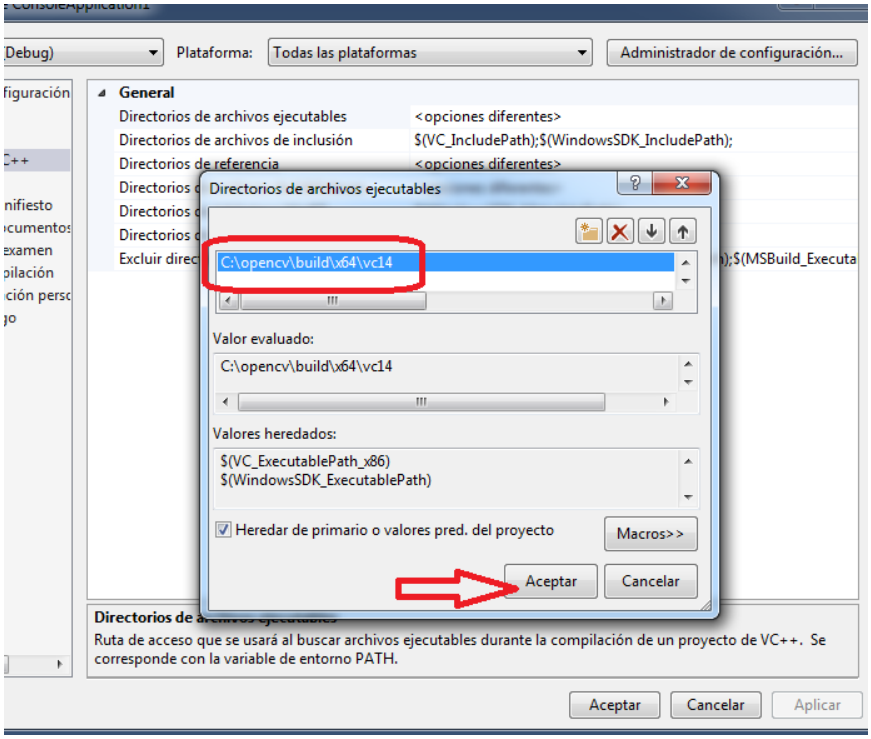


Figura 5: *OpenCV 5.*

El resto de procesos es muy similar al realizado. Más abajo en *Directorios de archivos de bibliotecas* se realiza la misma acción pero con este directorio: *C:\opencv\build\x64\vc14\lib.* Una vez aplicado, debería quedar así:

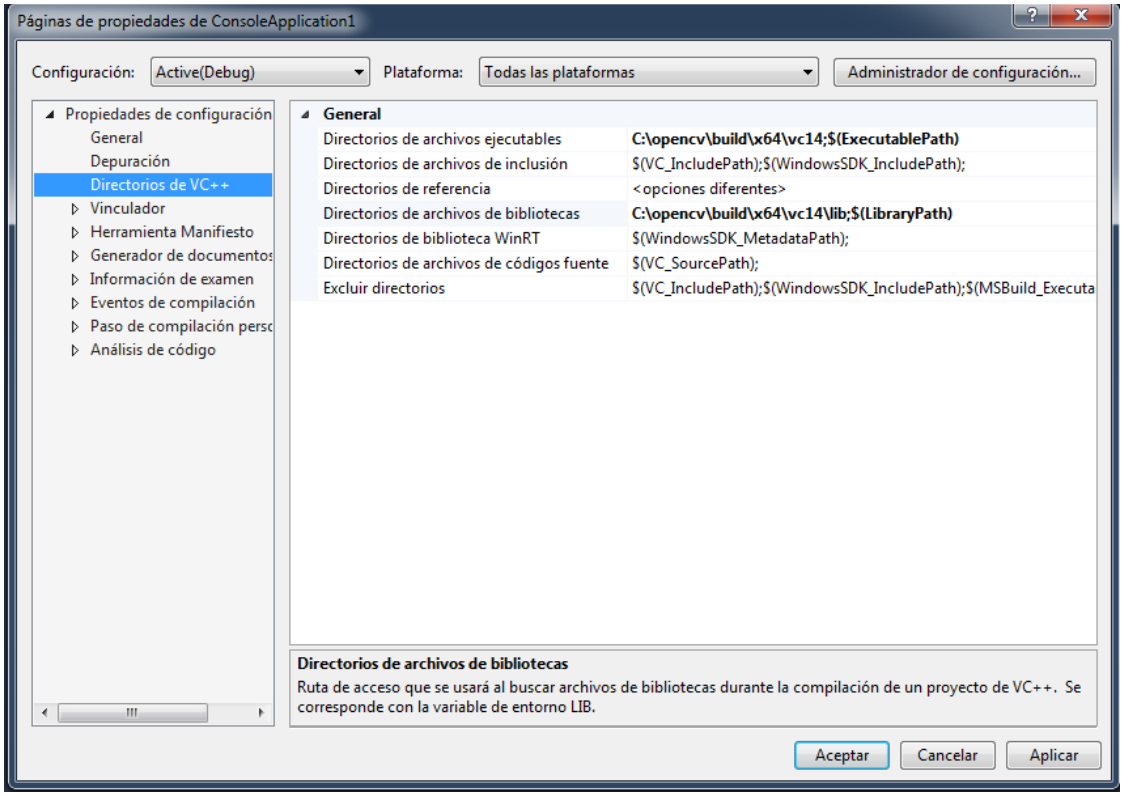


Figura 6: *OpenCV 6.*

Ahora, en el menú de la izquierda, justo debajo *de Directorios de VC++*, se encuentra *Vinculador*. Dentro de éste, abrimos la pestaña *Entrada*, y se selecciona *Dependencias adicionales*. En esta pestaña se añaden todos los archivos de extensión .lib de la carpeta lib que se añadió anteriormente. Como es la configuración para Debug (ventana de propiedades en la parte superior derecha), se añaden todos los archivos que terminen en d su nombre. En versiones anteriores de OpenCV había bastantes archivos, pero en esta versión solo hay uno: opencv_world310d.lib. Ésta es la única diferencia para configurar debug y release; en el caso de configurar release se añadirá: opencv_world310.lib.

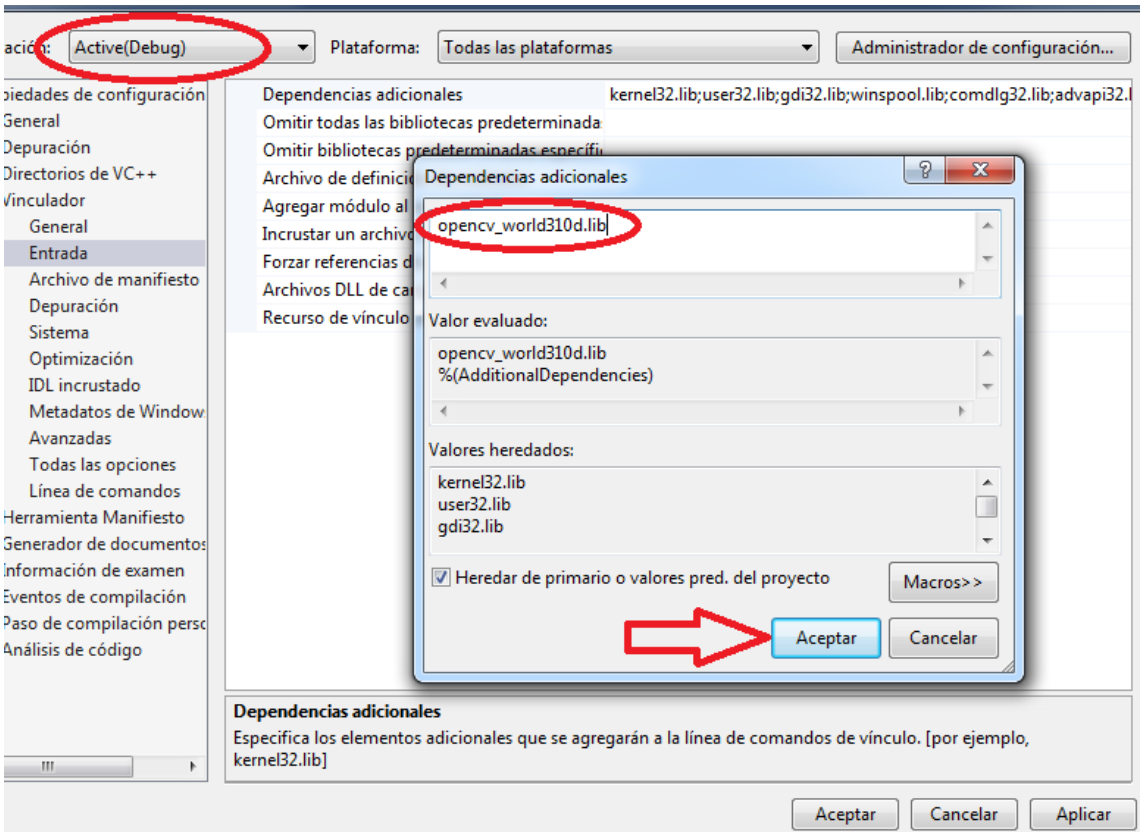


Figura 7: *OpenCV 7.*

A continuación, se crea el primer archivo del proyecto. En la ventana de *Explorador de soluciones*, haciendo click derecho en *Archivos de código fuente* se despliega un menú. Se hace click en *Agregar* y luego en *Nuevo elemento*.

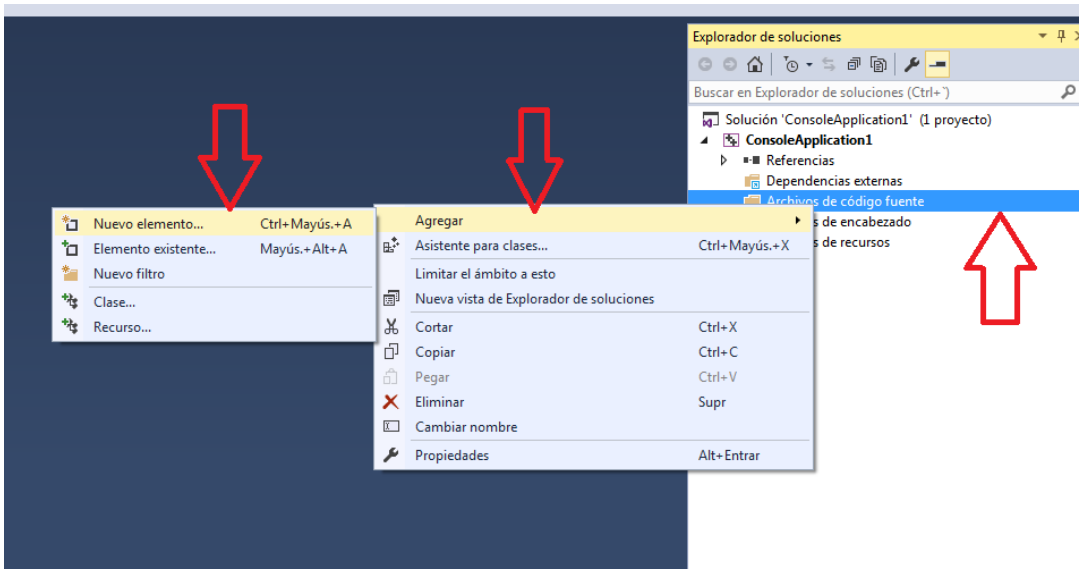


Figura 8: *OpenCV 8.*

La ventana que aparecerá a continuación nos indicará a la izquierda que se trata del lenguaje de programación *Visual C++*, y a la derecha mostrará dos opciones: *Archivo C++ (.cpp)* y *Archivo de encabezado (.h)*. Se elegirá la primera opción.

A este archivo se le llamará *main.cpp* (.cpp es la extensión de los archivos de código fuente; se llamará main porque significa "principal", que es justo la función que desempeñará este archivo: ser el archivo principal donde irá escrito el programa); y se crea de la siguiente manera:

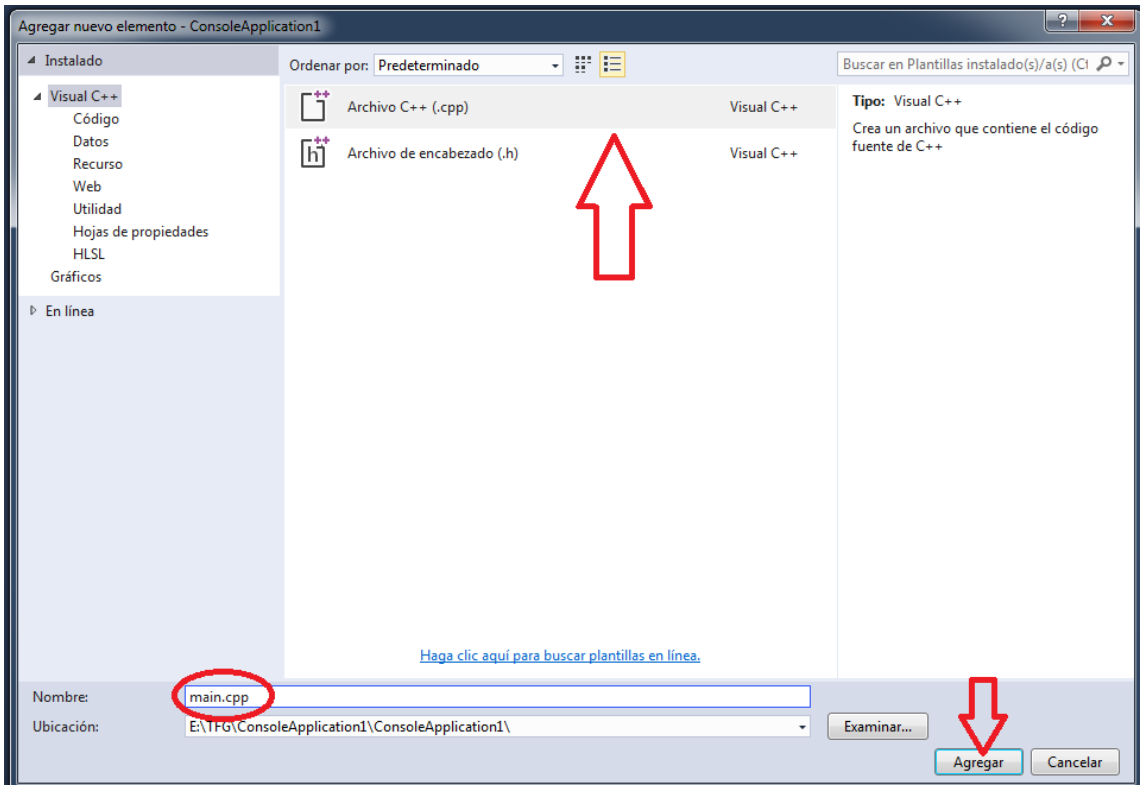


Figura 9: *OpenCV 9.*

Una vez hecho esto, el aspecto de la interfaz del proyecto es el siguiente:

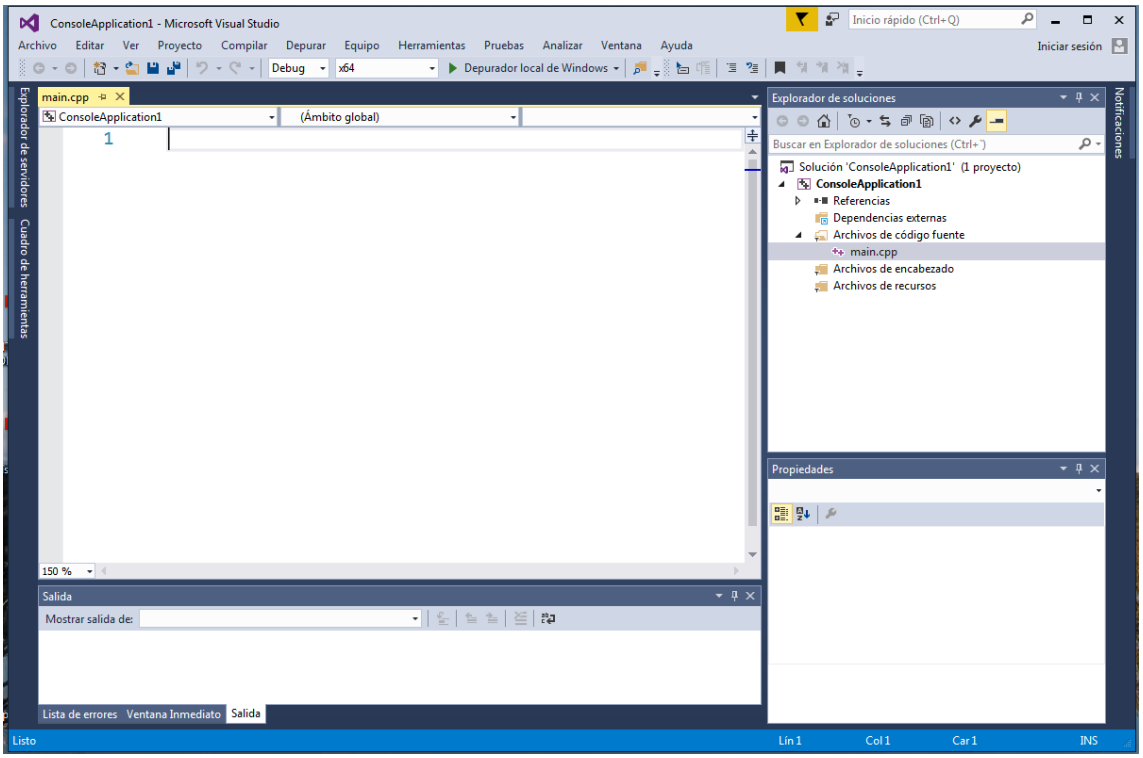


Figura 10: *OpenCV 10.*

Es importante, en la parte superior, escoger la configuración *x64*, ya que actualmente OpenCV sólo trae los archivos para 64 bits.

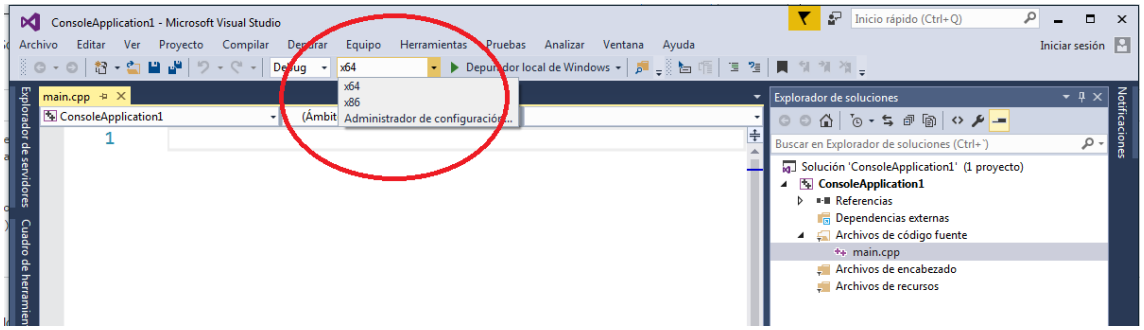


Figura 11: *OpenCV 11.*

Tan sólo falta una última cosa y ya estará listo para empezar a programar. En el *Explorador de soluciones*, sobre el nombre del proyecto, se da click a *Propiedades* (botón derecho) A la izquierda, ahora aparece una nueva pestaña: *C/C++.* En la pestaña *General*, se abre *Directorios de inclusión adicionales* y se añade este archivo: *C\opencv\build\include.*

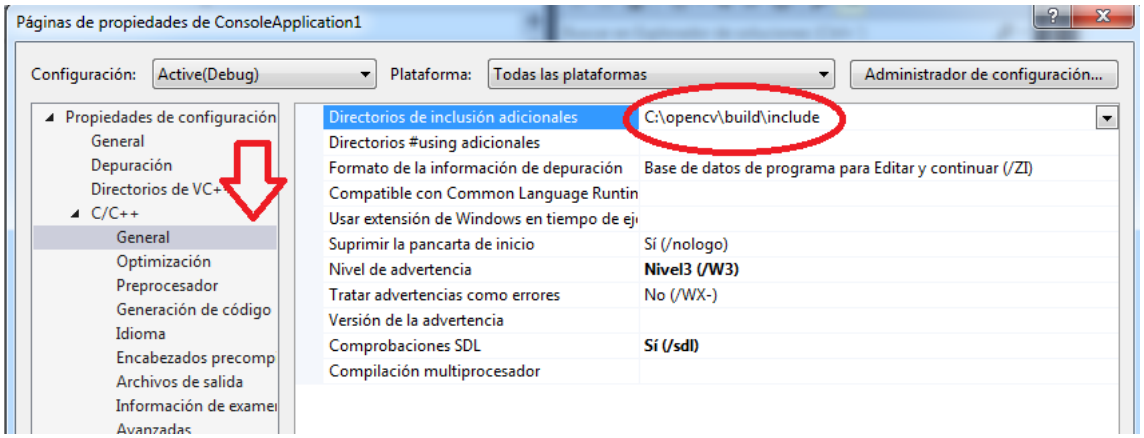


Figura 12: *OpenCV 12.*

Creación de una plantilla de OpenCV

Como cada vez que se quiera usar la librería OpenCV se debe realizar de nuevo todo el proceso descrito, se creará, a partir del proyecto actual, una plantilla que pueda ser reutilizada todas las veces que se quiera. Lo que se hace realmente, es guardar el proyecto creado con su configuración a modo de plantilla, guardada como un archivo, al cual se puede acceder al crear un proyecto nuevo.

Se seguirán los siguientes pasos para llegar a crear la plantilla: *Archivo / Exportar plantilla…*; ahora se desplegará una ventana llamada *Asistente para exportar plantillas*. Se selecciona *Plantilla de proyecto*, y abajo se escoge el proyecto del cual se quiere crear la plantilla. Se le da a "Siguiente" y, en la siguiente ventana, simplemente seleccionamos el nombre que se quiera para la plantilla y una breve descripción opcional, así como la ruta donde será guardada la plantilla creada.

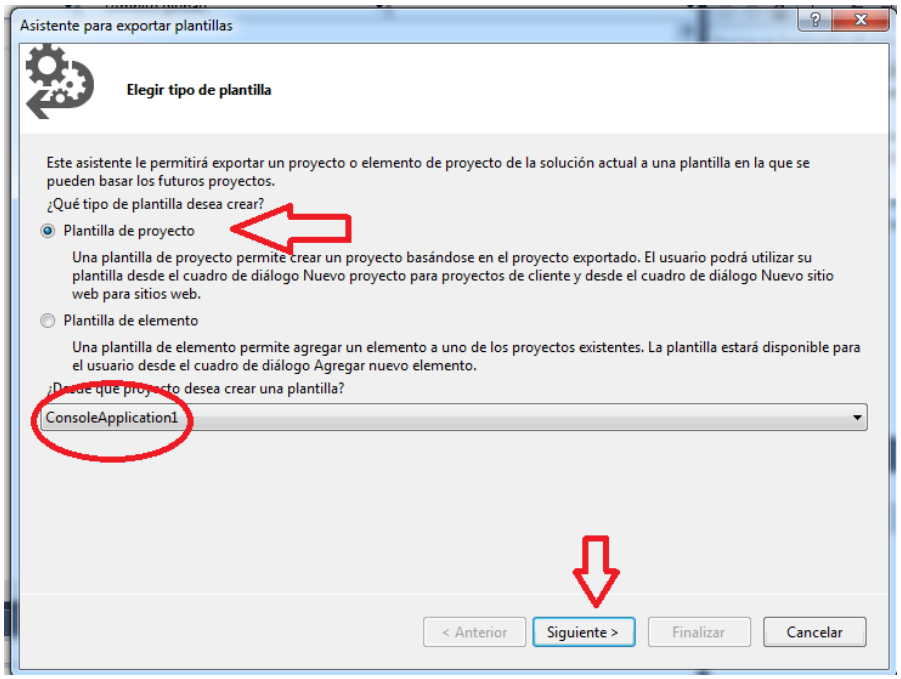


Figura 13: *OpenCV 13.*

Con todo esto, ya se puede comenzar a programar. Se crea un nuevo proyecto a partir de la plantilla de OpenCV siguiendo los pasos de creación de un nuevo proyecto, salvo que en esta ocasión, dentro de las posibles aplicaciones de Visual C++, aparecerá nuestra plantilla como opción.

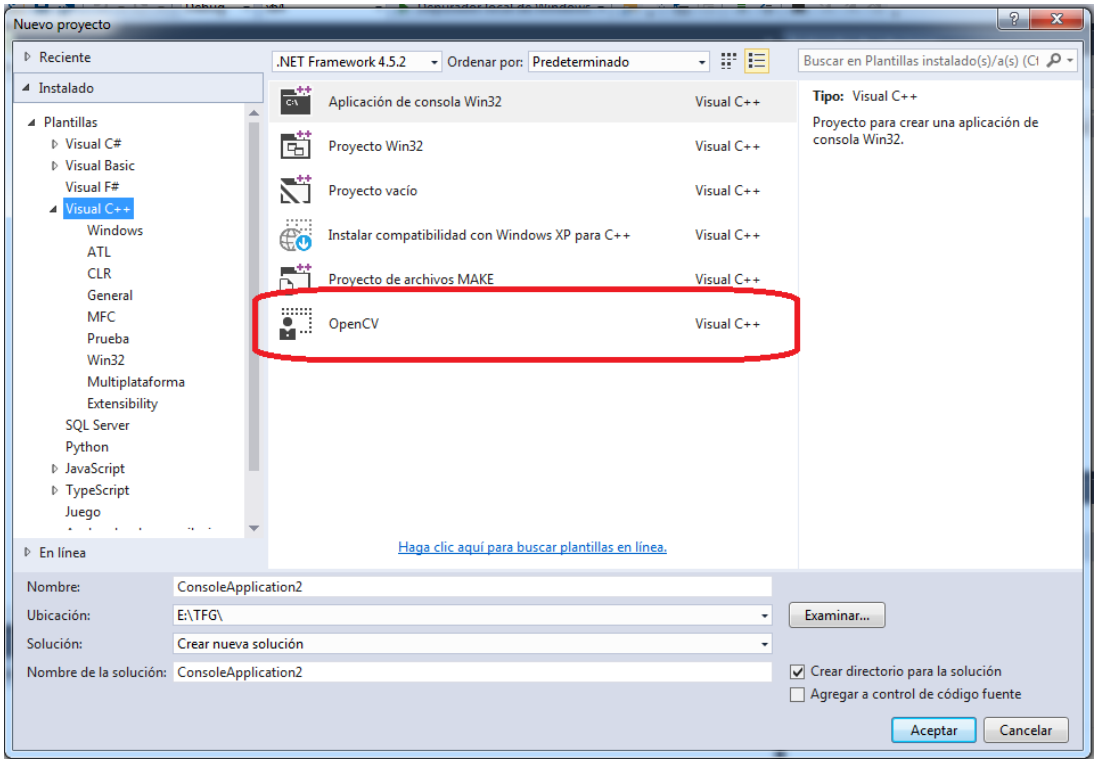


Figura 14: *OpenCV 14.*

ANEXO IV. MANUAL DE USUARIO

Para completar el programa, se ha incluido una interfaz gráfica para facilitar su utilización. Al iniciar, saldría por pantalla la *Figura 1*:

Figura 1: *Inicio interfaz.*

La interfaz sólo tiene dos botones: Seleccionar Circuito y Extraer. Con la primera opción se accede al fichero donde se tengan guardadas las imágenes de los circuitos. Cuando se elija la imagen, se abre y aparecerá la ruta a dicha imagen en el espacio rectangular (*Figura 2*).

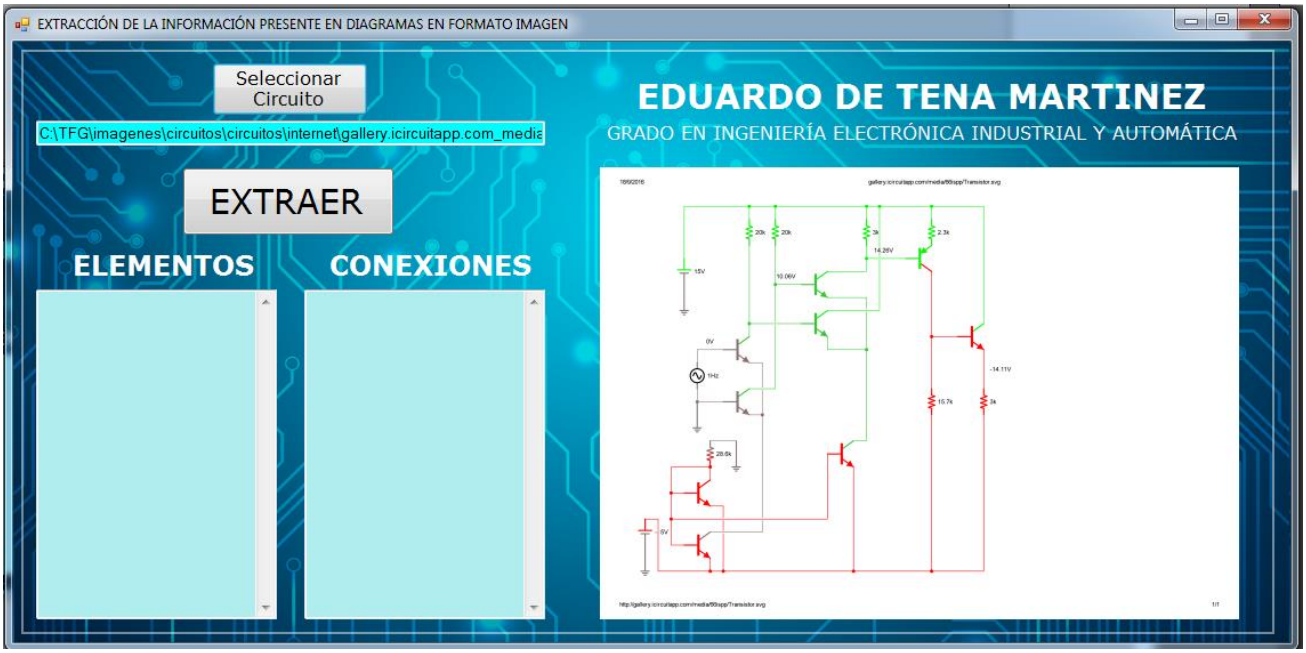


Figura 2: *Seleccionar circuito.*

Una vez seleccionado el circuito, se procede a extraer la información pulsando el botón Extraer (*Figura 3*).

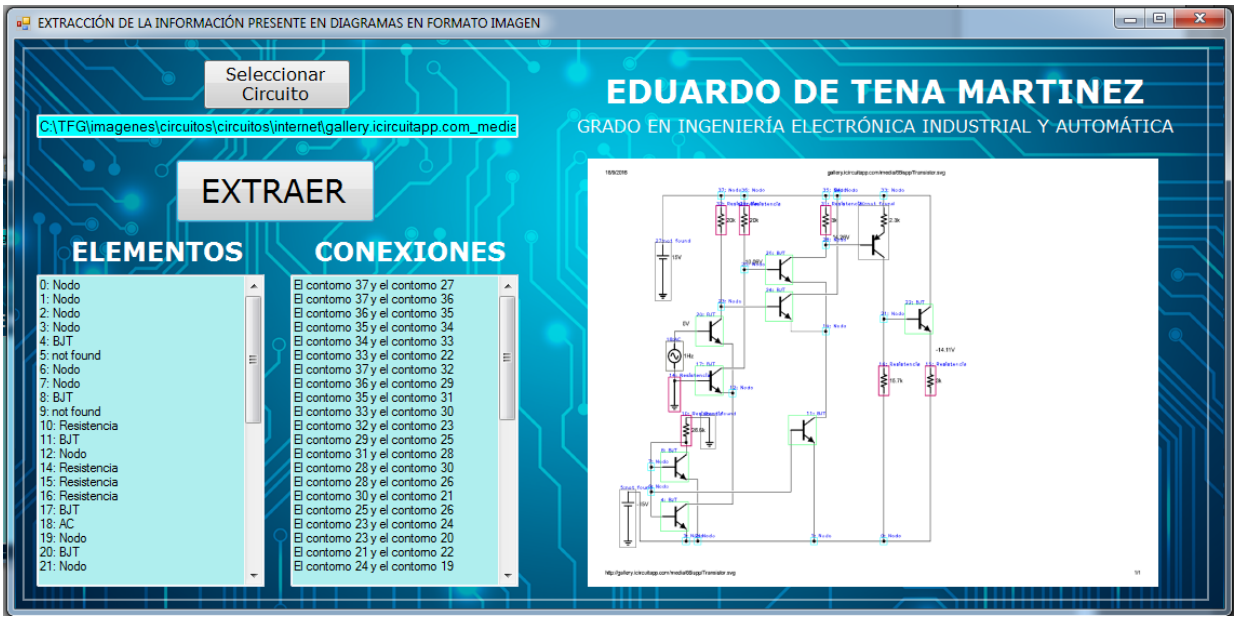


Figura 3: *Extraer.*

Es ahora cuando se ejecuta el programa principal y muestra por pantalla toda la información del circuito: en el espacio de la izquierda mostrará la lista de elementos, y en el espacio de la derecha las conexiones. También aparece la imagen original del circuito con todos los nombres de los elementos, la cual se expande haciendo doble click (*Figura 4*).

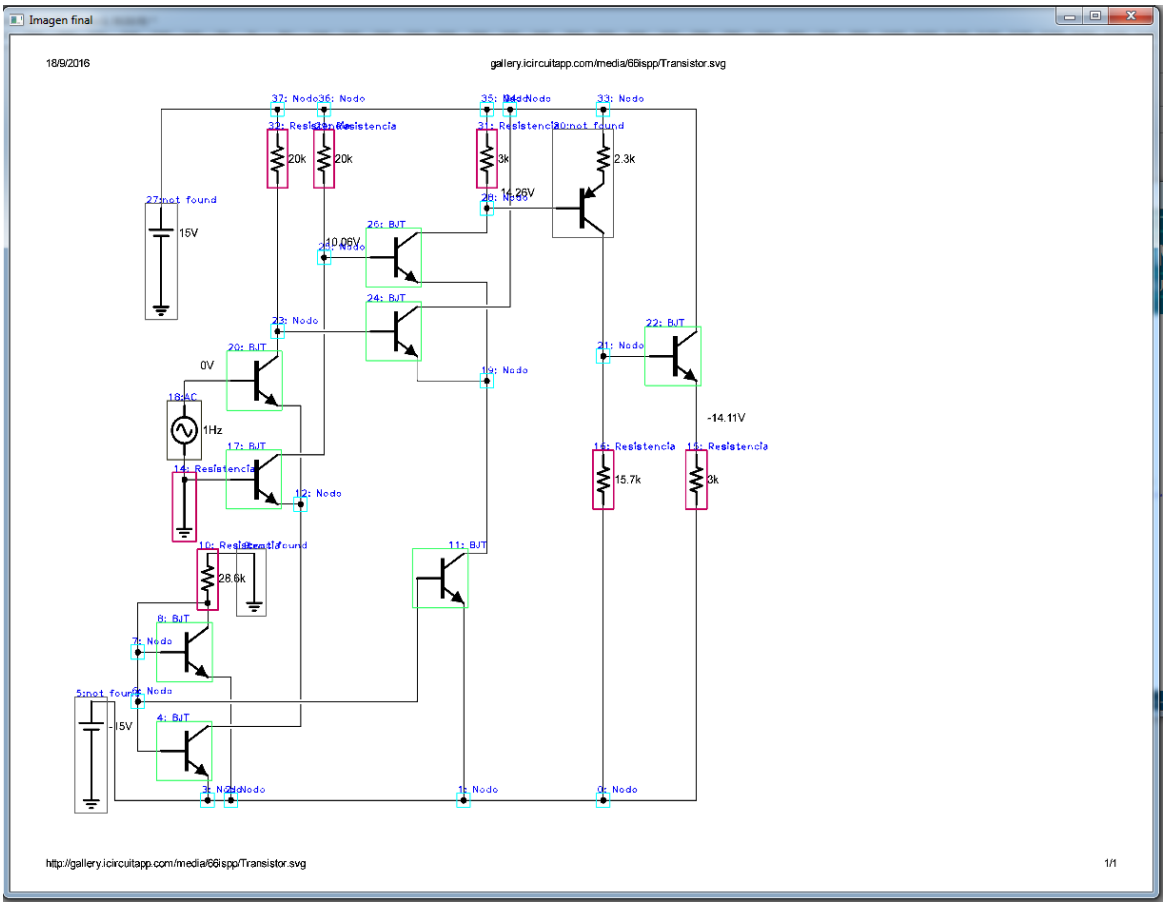


Figura 4: *Resultado final del programa.*

ANEXO V. EJEMPLOS DE CIRCUITOS PROCESADOS

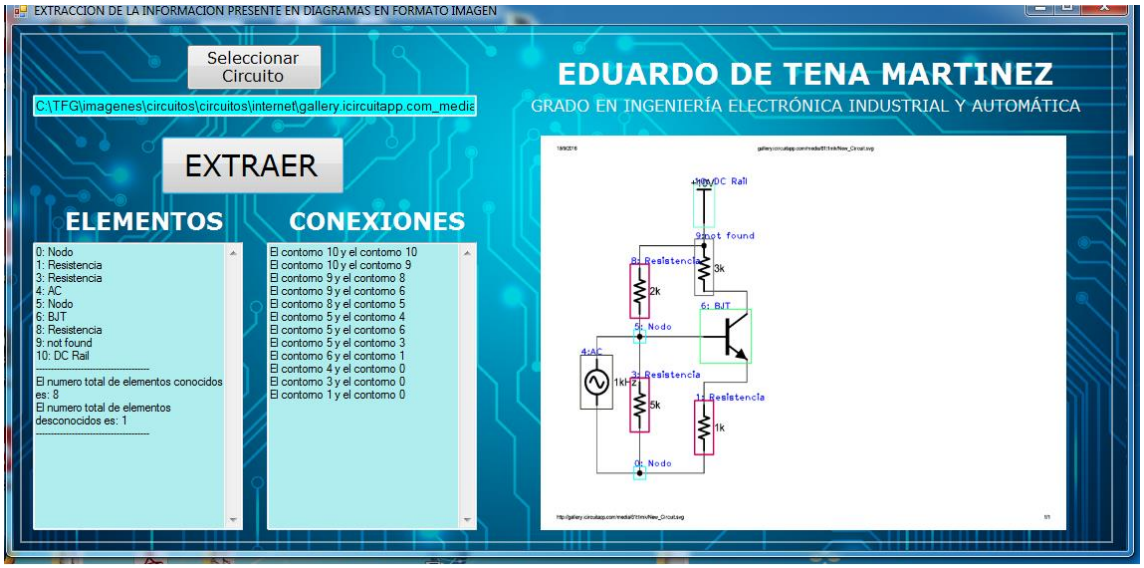


Figura 1: *Ejemplo 1.*

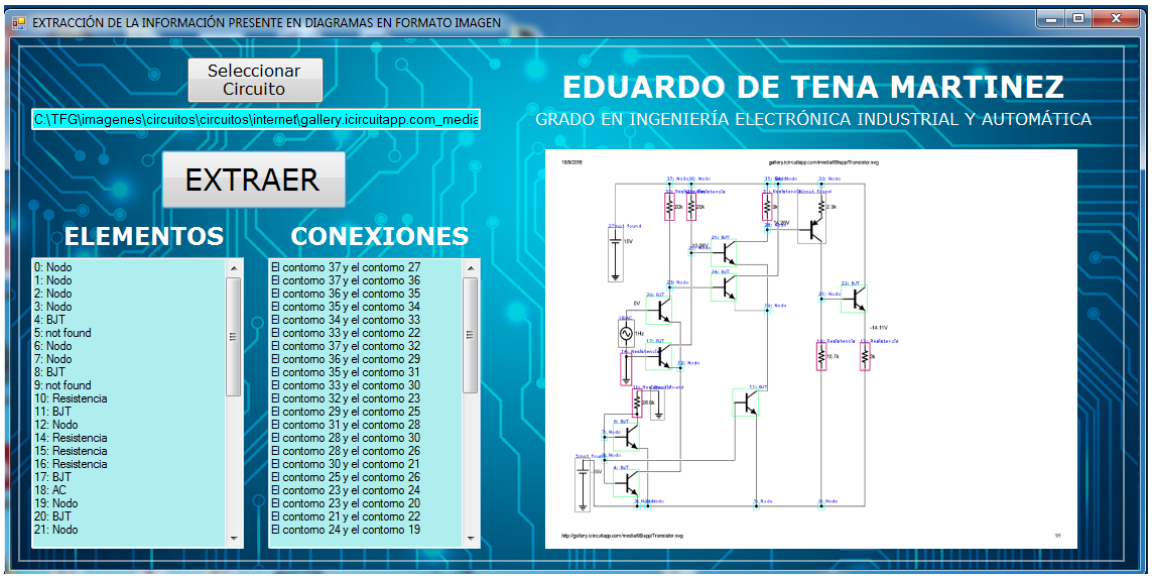


Figura 2: *Ejemplo 2.*

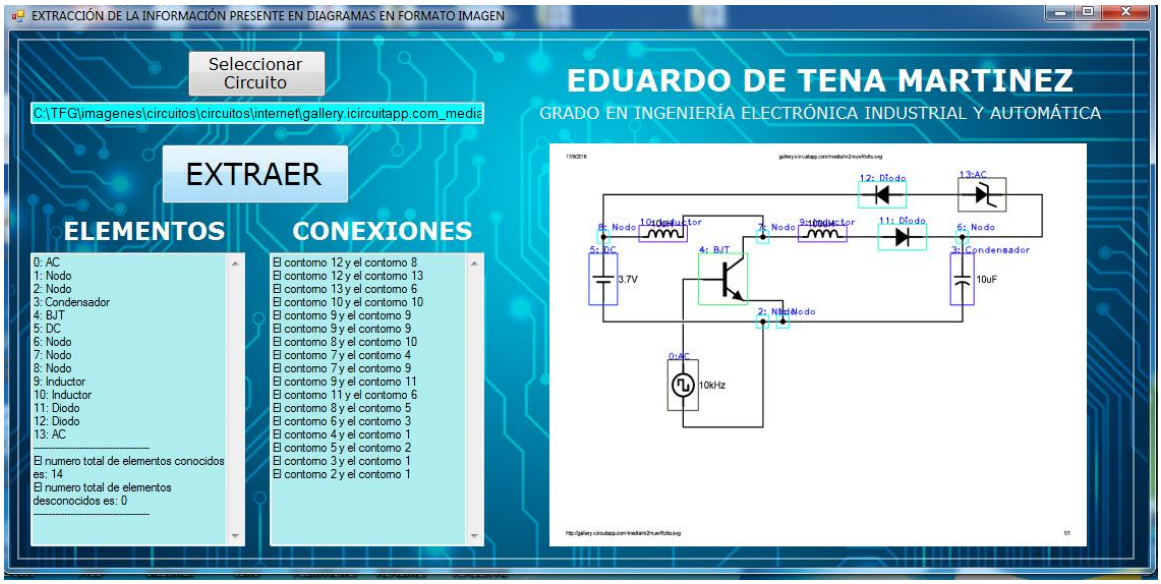


Figura 3: *Ejemplo 3.*

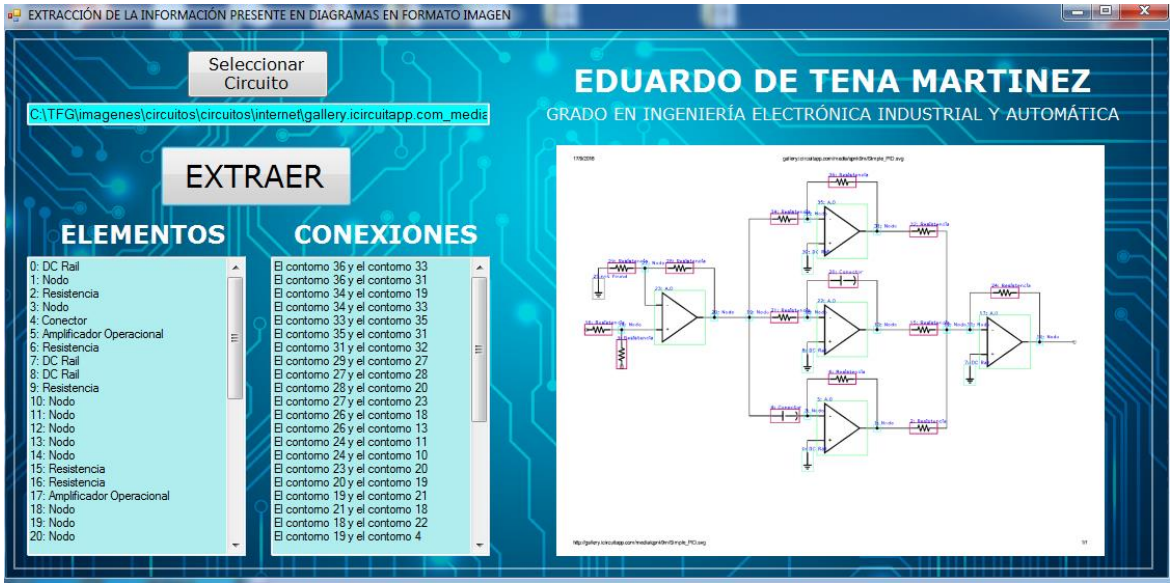


Figura 4: *Ejemplo 4.*

EXTRACCIÓN DE LA INFORMACIÓN PRESENTE EN DIAGRAMAS EN FORMATO IMAGEN